# CHAPTER ONE

## INTRODUCTION

### BACKGROUND INFORMATION

File sharing is the practice of sharing files among different users across a network platform. Common forms of file sharing are FTP (File Transfer Protocol) model and P2P (Peer to Peer as the basic among network type) file sharing network. Another common form of sharing files over the internet is for a user to upload files to and allow other users to download them from the website. There are lots of factors to consider when develop such website (this form works well on a client server based system).

Users of an online file sharing website who use features like upload, download, share, search etc. would want a website that is very interactive and fast and also a welcoming level of security check and not annoying with a lot of post backs and flashing screens. Another fact is that files should be retrievable whenever it is needed to. The transfer of file on a website platform needs a security focus because of the activity of the scripts writers who develop scripts to intercept and capture any file being transferred via a network to access a copy of it.

Encryption and Decryption mechanism is one of the computing ways of handing security in communication technology and this way has been favorable and productive since the first time of use. This cryptographic method of security handing helps to render information or files to be communicated on the network (internet and other electronic based communication platforms) meaningless to unauthorized personnel and as time passes by, various cryptographic algorithms has been developed to suite the strength of cryptographic method of computer security check.

Secured File Transfer System using RSA cryptography is an application that allows users to share files among one another and it provides a logical platform where files to be shared is being encrypted before sending and decrypted at the receiving end, the cryptographic method adopted is known as RSA (Rivest-Shamir-Adleman) cryptographic algorithm. The acronym for the algorithm was generated form the names of its invertors Ron Rivest, Adi Shamir and Leonard Adleman.

### 1.2 PROBLEM DEFINITION

Many applications either web application of a native application that adopt file transfer system mostly defines no room for security consideration of the file instead, the security is placed only on the application accessibility, this action yields the following problems:

1. The file being transferred has no security mechanism defines on it during the transfer process (while in motion of transfer).
2. The normal web based file folder is useless if there are other ways of interception without direct access to the file.
3. The file to be transferred is sometimes restricted and limited to specific sizes and types.
4. Users of an online file sharing websites who use features like share, send, upload, and download would prefer not only an interactive platform but also platform that support secured.
5. Two places where one can store uploaded file is database and server folder, if the file content is not secured and the database or server folder is accessed by intruders, then user’s data integrity is incomplete.

### 1.3 AIMS AND OBJECTIVES

This project aim to design a secured file transfer system using a cryptographic algorithm known as RSA to secure user’s file during sending and at the receiving end. Other system objectives are:

1. To review related works of the methods of file transfer with the adoption of cryptography as a new security measure.
2. To design an application that uses RSA algorithm to encrypt and decrypt files.
3. To implement a file transfer system that restrict user’s ability to directly access a file during receiving unless with the user’s permission.
4. To evaluate the effectiveness of using cryptography in the securing files.

### 1.4 PROJECT SCOPE AND LIMITATION

This project is a web based application platform that its design was achieved with the use of PHP, RSA cryptographic algorithm, bootstrap plugin, and MySQL database server. It covers the following scope:

1. **The Signup and Login Section:** a platform where users can create account into the website, and also login to their accounts.
2. **The Log Dashboard:** a dashboard that projects the list for files being transferred by a particular user.
3. **Profile Management:** a platform for users to manage their personal and login info.
4. **Transfer Dashboard:** a platform for users to open and upload a file for transfer, select user to transfer file to and perform the transferring of file.
5. **Reception Dashboard:** a platform for user to view list of received files and decrypt and open the directory of the decrypted file.

This project is tested on a local server and multiple browsers is opened to test the effectiveness of the web application, it does not support a live server visits instead, anther platform is to test it on a local network, or Wi-Fi network where the application will be executed on a server and other client system will access the system through the server’s IP address on a web browser which requires the knowledge of a networking expert to handle.

### 1.5 DEFINITION OF TERMS

* 1. **Client-Server**: a computer network in which one centralized, powerful computer (called the server) is a hub to which many less powerful personal computers or workstations (called clients) are connected.
  2. **Cryptography:** the enciphering and deciphering of messages in secret code or cipher also: the computerized encoding and decoding of information.
  3. **Cryptographic Key:** a string of bits used by a cryptographic algorithm to transform plain text into cipher text or vice versa.
  4. **Data Security:** refers to protective digital privacy measures that are applied to prevent unauthorized access to computers, databases and websites.
  5. **Decryption:** Any procedure used in cryptography to convert ciphertext (encrypted data) into plaintext.
  6. **Encryption:** any procedure used in cryptography to convert plaintext into ciphertext (encrypted message) in order to prevent any but the intended recipient from reading that data.
  7. **File Transfer:** a generic term for the act of transmitting files over a computer network like the Internet. There are numerous ways and protocols to transfer files over a network.
  8. **File Transfer Protocol:** a standard network protocol used to transfer computer files between a client and server on a computer network.
  9. **Key Space:** the set of all possible permutations of a [keys](https://en.wikipedia.org/wiki/Key_%28cryptography%29).
  10. **Peer-to-Peer:** a distributed application architecture that partitions tasks or workloads between peers.
  11. **Security Mechanisms:** a method, tool, or procedure for enforcing a security policy.
  12. **Symmetric Key:** A cryptography system in which both parties have the same encryption key, as in secret key cryptography.

# CHAPTER TWO

## LITERATURE REVIEW

### BRIEF HISTORY OF FILE TRANFER SYSTEM

FTP is built on a client-server model architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS). SSH File Transfer Protocol (SFTP) is sometimes also used instead, but is technologically different.

The first FTP client applications were command-line programs developed before operating systems had graphical user interfaces, and are still shipped with most Windows, UNIX, and Linux operating systems. Many FTP clients and automation utilities have since been developed for desktops, servers, mobile devices, and hardware, and FTP has been incorporated into productivity applications, such as web page editors (File Transfer Protocol, 2015).

Files were first exchanged on removable media. Computers were able to access remote files using filesystem mounting, bulletin board systems (1978), Usenet (1979), and FTP servers (1985). Internet Relay Chat (1988) and Hotline (1997) enabled users to communicate remotely through chat and to exchange files. The mp3 encoding, which was standardized in 1991 and which substantially reduced the size of audio files, grew to widespread use in the late 1990s. In 1998, MP3.com and Audiogalaxy were established, the Digital Millennium Copyright Act was unanimously passed, and the first mp3 player devices were launched.

In June 1999, Napster was released as an unstructured centralized peer-to-peer system, requiring a central server for indexing and peer discovery. It is generally credited as being the first peer-to-peer file sharing system.

Gnutella, eDonkey2000, and Freenet were released in 2000, as MP3.com and Napster were facing litigation. Gnutella, released in March, was the first decentralized file sharing network. In the gnutella network, all connecting software was considered equal, and therefore the network had no central point of failure. In July, Freenet was released and became the first anonymity network. In September the eDonkey2000 client and server software was released.

In 2001, Kazaa and Poisoned for the Mac was released. Its FastTrack network was distributed, though unlike gnutella, it assigned more traffic to 'supernodes' to increase routing efficiency. The network was proprietary and encrypted, and the Kazaa team made substantial efforts to keep other clients such as Morpheus off of the FastTrack network.

In July 2001, Napster was sued by several recording companies and lost in A&M Records, Inc. v. Napster, Inc. In the case of Napster, it has been ruled that an online service provider could not use the "transitory network transmission" safe harbor in the DMCA if they had control of the network with a server.

Shortly after its loss in court, Napster was shut down to comply with a court order. This drove users to other P2P applications and file sharing continued its growth. The Audiogalaxy Satellite client grew in popularity, and the LimeWire client and BitTorrent protocol were released. Until its decline in 2004, Kazaa was the most popular file sharing program despite bundled malware and legal battles in the Netherlands, Australia, and the United States. In 2002, a Tokyo district court ruling shut down File Rogue, and the Recording Industry Association of America (RIAA) filed a lawsuit that effectively shut down Audiogalaxy.

Demonstrators protesting The Pirate Bay raid, 2006.

From 2002 through 2003, a number of BitTorrent services were established, including Suprnova.org, isoHunt, TorrentSpy, and The Pirate Bay. In 2002, the RIAA was filing lawsuits against Kazaa users. As a result of such lawsuits, many universities added file sharing regulations in their school administrative codes (though some students managed to circumvent them during after school hours). With the shutdown of eDonkey in 2005, eMule became the dominant client of the eDonkey network. In 2006, police raids took down the Razorback2 eDonkey server and temporarily took down The Pirate Bay.

In 2009, the Pirate Bay trial ended in a guilty verdict for the primary founders of the tracker. The decision was appealed, leading to a second guilty verdict in November 2010. In October 2010, Limewire was forced to shut down following a court order in Arista Records LLC v. Lime Group LLC but the gnutella network remains active through open source clients like Frostwire and gtk-gnutella. Furthermore, multi-protocol file sharing software such as MLDonkey and Shareaza adapted in order to support all the major file sharing protocols, so users no longer had to install and configure multiple file sharing programs.

On January 19, 2012, the United States Department of Justice shut down the popular domain of Megaupload (established 2005). The file sharing site has claimed to have over 50,000,000 people a day. Kim Dotcom (formerly Kim Schmitz) was arrested in New Zealand and is awaiting extradition. The case involving the downfall of the world's largest and most popular file sharing site was not well received, with hacker group Anonymous bringing down several sites associated with the take-down. In the following days, other file sharing sites began to cease services; Filesonic blocked public downloads on January 22, with Fileserve following suit on January 23 (File sharing, 2013).

### HISTORY OF CRYPTOGRAPHY

The first known evidence of the use of cryptography (in some form) was found in an inscription carved around 1900 BC, in the main chamber of the tomb of the nobleman Khnumhotep II, in Egypt. The scribe used some unusual hieroglyphic symbols here and there in place of more ordinary ones. The purpose was not to hide the message but perhaps to change its form in a way which would make it appear dignified. Though the inscription was not a form of secret writing, but incorporated some sort of transformation of the original text, and is the oldest known text to do so. Evidence of some use of cryptography has been seen in most major early civilizations. "Arthshashtra", a classic work on statecraft written by Kautalya, describes the espionage service in India and mentions giving assignments to spies in "secret writing" - sounds like an ancient version of James Bond?

Fast forwarding to around 100 BC, Julius Caesar was known to use a form of encryption to convey secret messages to his army generals posted in the war front. This substitution cipher, known as Caesar cipher, is perhaps the most mentioned historic cipher in academic literature. (A cipher is an algorithm used for encryption or decryption.) In a substitution cipher, each character of the plain text (plain text is the message which has to be encrypted) is substituted by another character to form the cipher text (cipher text is the encrypted message). The variant used by Caesar was a shift by 3 cipher. Each character was shifted by 3 places, so the character 'A' was replaced by 'D', 'B' was replaced by 'E', and so on. The characters would wrap around at the end, so 'X' would be replaced by 'A'.

It is easy to see that such ciphers depend on the secrecy of the system and not on the encryption key. Once the system is known, these encrypted messages can easily be decrypted. In fact, substitution ciphers can be broken by using the frequency of letters in the language.

During the 16th century, Vigenere designed a cipher that was supposedly the first cipher which used an encryption key. In one of his ciphers, the encryption key was repeated multiple times spanning the entire message, and then the cipher text was produced by adding the message character with the key character modulo 26. (Modulo, or mod, is a mathematical expression in which you calculate the remainder of a division when one number is divided by another.) As with the Caesar cipher, Vigenere's cipher can also easily be broken; however, Vigenere's cipher brought the very idea of introducing encryption keys into the picture, though it was poorly executed. Comparing this to Caesar cipher, the secrecy of the message depends on the secrecy of the encryption key, rather than the secrecy of the system.

At the start of the 19th century when everything became electric, Hebern designed an electro-mechanical contraption which was called the Hebern rotor machine. It uses a single rotor, in which the secret key is embedded in a rotating disc. The key encoded a substitution table and each key press from the keyboard resulted in the output of cipher text. This also rotated the disc by one notch and a different table would then be used for the next plain text character. This was again broken by using letter frequencies.

The Engima machine was invented by German engineer Arthur Scherbius at the end of World War I, and was heavily used by the German forces during the Second World War. The Enigma machine used 3 or 4 or even more rotors. The rotors rotate at different rates as you type on the keyboard and output appropriate letters of cipher text. In this case the key was the initial setting of the rotors. The Enigma machine's cipher was eventually broken by Poland and the technology was later transferred to the British cryptographers who designed a means for obtaining the daily key.

Up to the Second World War, most of the work on cryptography was for military purposes, usually used to hide secret military information. However, cryptography attracted commercial attention post-war, with businesses trying to secure their data from competitors.

In the early 1970's, IBM realized that their customers were demanding some form of encryption, so they formed a "crypto group" headed by Horst-Feistel. They designed a cipher called Lucifer. In 1973, the Nation Bureau of Standards (now called NIST) in the US put out a request for proposals for a block cipher which would become a national standard. They had obviously realized that they were buying a lot of commercial products without any good crypto support. Lucifer was eventually accepted and was called DES or the Data Encryption Standard. In 1997, and in the following years, DES was broken by an exhaustive search attack. The main problem with DES was the small size of the encryption key. As computing power increased it became easy to brute force all different combinations of the key to obtain a possible plain text message.

In 1997, NIST again put out a request for proposal for a new block cipher. It received 50 submissions. In 2000, it accepted Rijndael, and christened it as AES or the Advanced Encryption Standard.

### SOME COMMON EXAMPLES OF CRYPTOGRAPHIC ALGORITHM

1. **Triple DES:** Triple DES was designed to replace the original Data Encryption Standard (DES) algorithm, which hackers eventually learned to defeat with relative ease. At one time, Triple DES was the recommended standard and the most widely used symmetric algorithm in the industry. Triple DES uses three individual keys with 56 bits each. The total key length adds up to 168 bits, but experts would argue that 112-bits in key strength is more like it.

Despite slowly being phased out, Triple DES still manages to make a dependable hardware encryption solution for financial services and other industries.

1. Twofish: Computer security expert Bruce Schneier is the mastermind behind Blowfish and its successor Twofish. Keys used in this algorithm may be up to 256 bits in length and as a symmetric technique, only one key is needed.

Twofish is regarded as one of the fastest of its kind, and ideal for use in both hardware and software environments. Like Blowfish, Twofish is freely available to anyone who wants to use it. As a result, you’ll find it bundled in encryption programs such as PhotoEncrypt, GPG, and the popular open source software TrueCrypt.

1. AES: The Advanced Encryption Standard (AES) is the algorithm trusted as the standard by the U.S. Government and numerous organizations. Although it is extremely efficient in 128-bit form, AES also uses keys of 192 and 256 bits for heavy duty encryption purposes.

AES is largely considered impervious to all attacks, with the exception of brute force, which attempts to decipher messages using all possible combinations in the 128, 192, or 256-bit cipher. Still, security experts believe that AES will eventually be hailed the de facto standard for encrypting data in the private sector.

### RSA (RIVEST-SHAMIR-ADLEMAN) CRYPTOGRAPHY

RSA is a public-key encryption algorithm and the standard for encrypting data sent over the internet. It also happens to be one of the methods used in our PGP and GPG programs. Unlike Triple DES, RSA is considered an asymmetric algorithm due to its use of a pair of keys. You’ve got your public key, which is what we use to encrypt our message, and a private key to decrypt it. The result of RSA encryption is a huge batch of mumbo jumbo that takes attackers quite a bit of time and processing power to break.

RSA was first described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- factoring -- is considered infeasible due to the time it would take even using today’s super computers (RSA algorithm (Rivest-Shamir-Adleman) , 2006).

The public and the private key-generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, p and q, are generated using the Rabin-Miller primality test algorithm. A modulus n is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus n, and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The e figure doesn't have to be a secretly selected prime number as the public key is shared with everyone. The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n (RSA algorithm (Rivest-Shamir-Adleman) , 2006).

### RELATED WORKS

Margaret Rouse posted an article on <http://www.techtarget.com> which focus on the popularity of RSA algorithm the content of the article says that: RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- factoring -- is considered infeasible due to the time it would take even using today’s super computers.

The public and the private key-generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, p and q, are generated using the Rabin-Miller primality test algorithm. A modulus n is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus n, and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The e figure does not have to be a secretly selected prime number as the public key is shared with everyone. The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n (Margaret, 2013).

David wells wrote a book titled “Prime Number Hide-and-Seek: How the RSA Cipher Works”: The RSA cipher is a fascinating example of how some of the most abstract mathematical subjects find applications in the real world. Few are the mathematicians who study creatures like the prime numbers with the hope or even desire for their discoveries to be useful outside of their own domain. But every now and then that is exactly what happens. This text explains the mathematics behind RSA -- how and why it works. The intended audience is just about anyone who is interested in the topic and who can remember a few basic facts from algebra: what a variable is, the difference between a prime number and a composite number, and the like (Wells, 1989).

# CHAPTER THREE

## RESEARCH AND DESIGN METHODOLOGY

### MATERIALS AND METHODS (DIAGRAM)

This project research and design mostly took place online where pictures, ebooks, and other materials related to cryptographic algorithms was downloaded to develop the project. The project also uses PHP script and other web design scripts and a RSA cryptographic class will be created as a plugins to be called as a method of comparison in the website, MySQL for the database design and communication. The figures below shows the input-process-output (IPO) diagram of the cryptographic section with RSA cryptography:

**(Sender)**

Get User’s uploaded file

Perform RSA encryption to the byte content of the file from the sender, or decryption of the encrypted file to the receiver

**(Receiver)**

Download decrypted file being received

**Figure 3.1:** The IPO diagram of the file transfer process

The method adopted for the design is the use of a web based posting platform which is achieved using some of the web development programming tools, and the application provides a section for user to setup an encryption key and the ability of the web application to use the key for file transfer process.

### MATERIALS USED

During the development of this project, information were gathered and some materials on cryptographic algorithm were downloaded and studied from the internet. The contents that constitutes majority of the downloaded materials includes, ebooks, algorithms PHP format, and blogs on webpages. Some of the material sources includes:

1. RSA Encryption <http://mathworld.wolfram.com/RSAEncryption.html>
2. RSA algorithm (Rivest-Shamir-Adleman) <http://whatis.techtarget.com/glossary/Network-Security/rsa>
3. PHP RSA encryption and decryption <http://www.osd.net/blog/web-development/php/php-rsa-encryption-and-decryption/>
4. Simple RSA public key encryption algorithm implementation. <file:///E:/2016/Tunde%27s%20Frnd/Code/Simple%20RSA%20public%20key%20encryption%20algorithm%20implementation.%20%20%20RSA%C2%A0%C2%AB%C2%A0Security%C2%A0%C2%AB%C2%A0Java.htm>

### METHODS

Web application platform was chosen to realize the application design and the design tools includes the following:

**3.2.1 PHP:**

PHP is a programming language for building dynamic, interactive Web sites. As a general rule, PHP programs run on a Web server, and serve Web pages to visitors on request. One of the key features of PHP is that you can embed PHP code within HTML Web pages, making it very easy for you to create dynamic content quickly. It is deep enough to run the largest social network (Facebook),It is also easy enough to be a beginner's first server side language.

**3.2.2 MYSQL:**

MySQL is used as the database server engine to design the web application database. This provides environment to create a database and the tables in it. This server stores user’s personal details, security details, and user’s transferred files.

### 3.2.3 THE DATA COLLECTION PROCESSES

The approach to establish a research can be Primary; consisting of the native study of the topic under research and chiefly involves first direct methods in the form of questionnaires, surveys, interviews, observations and discussion groups, or Secondary; which requires external approach to research which usually involves perusal of mostly published works like researching through archives of public libraries, court rooms and published academic journals. Considering the nature of this project, the non-empirical approach was taken which is all about visiting existing archives of cryptographic solution approach and consult a meta-analysis on them to study how RSA cryptographic algorithm works.

### RESEARCH METHODOLOGY

In this section, a fact was acquired from observation of some of the common open file transfer support networked sharing app and websites, and discussions from online users. Many of these file transfer platform do not adopt the use of encryption and decryption on till of recent. During the research methodology RSA cryptography is rated among one of the highly secured standard of encryption and decryption mechanism that is widely used since the time of its creation; in fact PHP scripting platform adopt it to develop sha1, and md4 algorithm, it also appears that then reason why most of websites and other file transfer enabled platform do not include file encryption was the controversial properties and majorly file type and sizes that exists.

### 3.2.1 ALGORITHM

This project encourage the use of RSA cryptographic algorithm which is used to encrypt file to be transferred to a selected user from the sender page and decrypt it at the receiving end. Below is a PHP algorithm for RSA cryptography:

/\* RSA key initializer\*/

public $pubkey = '...public key here...';

public $privkey = '...private key here...';

/\* RSA encryption Method\*/

public function encrypt($data)

{

if (openssl\_public\_encrypt($data, $encrypted, $this->pubkey))

$data = base64\_encode($encrypted);

else

throw new Exception('Unable to encrypt data. Perhaps it is bigger than the key size?');

return $data;

}

/\* RSA key decryption method\*/

public function decrypt($data)

{

if (openssl\_private\_decrypt(base64\_decode($data), $decrypted, $this->privkey))

$data = $decrypted;

else

$data = '';

return $data;

}

### 3.2.2 DESIGN MECHANISMS

The application is designed to consist of three sections for realizing its entire structure and operations. The first section is the user’s data registration section where user signup to the website and the information collected during the process is stored in the user’s profile table. The second section is the file encryption and transfer section where user select a file to transfer; select the user and the destination of where the file is going and; click the transfer button which logically encrypt the file content before transferring it, section create a log in the transfer table so that user at the receiving end can view it from his or her own account. The third section is the receiver’s interface where received file can be viewed and downloaded after decryption.

### PROCESS

The file transfer system major activity is established on two stages. The first part is on the sending stage where the file to be transferred is encrypted using RSA algorithm. The second stage is the receiving stage were the sent file is decrypted using the RSA algorithm also at user’s response. The encryption and decryption process is as follows is made of the following steps:

1. Encryption process of file transfer.
   1. Upload a file to transfer.
   2. Select a user to transfer file to.
   3. Click the transfer button.
   4. The application logic calls the RSA encryption class and encrypt the file,
   5. The application send the encrypted file to user.
2. Decryption process of file transfer.
   1. View lists of received files
   2. Select a file to download
   3. Decrypt file to download.
   4. Download decrypted file.

### SYSTEM USE CASE

RSA Based Secured File Transfer Website

Website

System User

**Figure 3.2:** Use case diagram for transfer section

RSA Based Secured File Transfer Website

Website

System User

**Figure 3.3:** Use case diagram for receptor section.

### FLOWCHART OF SECURED FILE TRANSFER SYSTEM USING RSA

The website File Transfer System is designed for evaluating RSA cryptographic algorithm, consists of three basic interfaces, the landing interface which provides user the ability to sign up as new or login as existing user. The users main interface which is accessible after logging in form the landing interface; it shows the lists of sent and received file associated with the logged in user, provide a platform for user to decrypt received file and download decrypted, and also provide a platform to select a file from the users pc and send it to a selected user in an encrypted format using RSA. And the last interface for user to update personal profile consisting of full name, phone number and other sort, and also to perform update on security profile where security information can be modified. The flowchart below shows the detail of the application behavior:

Begin

Display the Home page

Visit the website URL on the local server

End

Click “Login” Button

Click “Sign up”

Supply signup details

Display User’s Dashboard Interface

View File transfer table logs

View received file

Logout

Supply Login Details

Verify Login Details Success status?

Yes

No

Show invalid message

Register User’s Signup Details

Decrypt and open received file

Fetch file to transfer

Select user to transfer file to

Encrypt and transfer file

**Figure** **3.5**: Secured File Transfer System Using RSA Algorithm

# CHAPTER FOUR

## 4.0 IMPLEMENTATION AND RESULTS

### 4.1 SYSTEM IMPLEMENTATION

File transfer system usage and settings in a new computer system or in a naked computer system requires some instructional procedures which appears to be simple and straight forward. To make it swift and detailed, these procedures will be listed and explain below but following the settings procedure at first list and the usage as next list.

The software settings requires the following steps:

Copy the file dmadict.zip to the required Computer System.

Unzip the file and install the wamp server that exists in the unzipped pack, or may decide to download the latest version for use and ensure that it is properly working.

Open the wamp server control panel 🡪Apache🡪Apache Modules🡪 Locate and mark apache rewrite module submenu.

Open the wamp server control panel again to access its www directory.

Copy secure file transfer system folder from the currently unzipped file to the already opened www directory.

Once again, open the wamp server control panel and access its phpmyadmin page.

Create a database with name **transfer.**

Import the sql file **transfer.sql** into the system and ensure that the operation is successful and the web application is ready for launching and usage.

The software usage requires just few steps which is listed below:

Start the wamp server whenever the system is turned on.

Open any of the system web browser.

Type the url [**http://localhost/**se**cured\_file\_transfer\_rsa/**](http://localhost/secured_file_transfer_rsa/%20) **/public** and the application is launched and the dictionary can be used.

The web application is working effectively and supports a suitable system requirement specification which is basically divided into the hardware and the software requirement as listed below:

### 4.2 HARDWARE REQUIREMENTS

The following are the some hardware specifications to support using the designed web application:

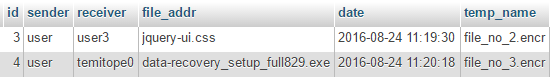
* 512MHz or Higher Intel Premium or AMD Processor.
* 256Mb Memory (RAM) or Higher.
* VGA 800 x 600, 256 color Minimum.
* Hard Disk Storage of 60 GB Minimum.

### 4.3 SOFTWARE REQUIREMENTS

* A 32 or 64bit Windows Operating System, Linux OS, Mac OS etc. that support the use of Apache Tomcat server.
* Reliable and licensed Antivirus software like Avast, AVG, or any system security shield.
* Web application testing and Mysql database local server (Xampp, Wamp, Lamp, etc.) with port set to 8080 connecting to system address (127.0.0.1) and not shared by any others app.
* Any typical web browser (Mozilla, Google Chrome, Operamini) that supports HTML5 execution.

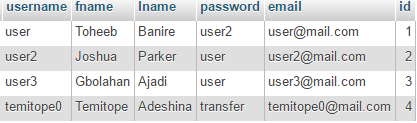
### 4.4 REFERRER DATABASE TABLE SCREENSHOTS

The website constitute two database tables for storing user’s signup information and file transfer information. The table that holds the file transfer information is labelled as **transfer.file\_trnasfer,** this table stores the file name, transfer generated names, sender, and receiver of the file, and also the date when the transfer is made. The screenshot in figure 4.1 below display some of the sample record stored in the file table during application testing process:



**Figure 4.1:** some of the transfer.file\_transfer table records.

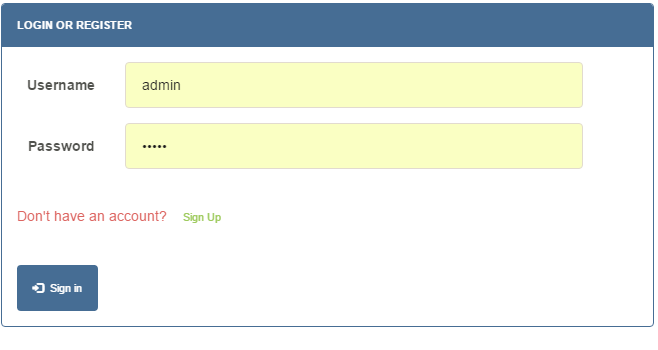
The table that stores users signup information is tagged as **transfer.users** which is made handles the storage of registered user’s username and password, first name, last name, and signup email. The screenshot in figure 4.2 below shows some of the record stored in the user’s table during testing.

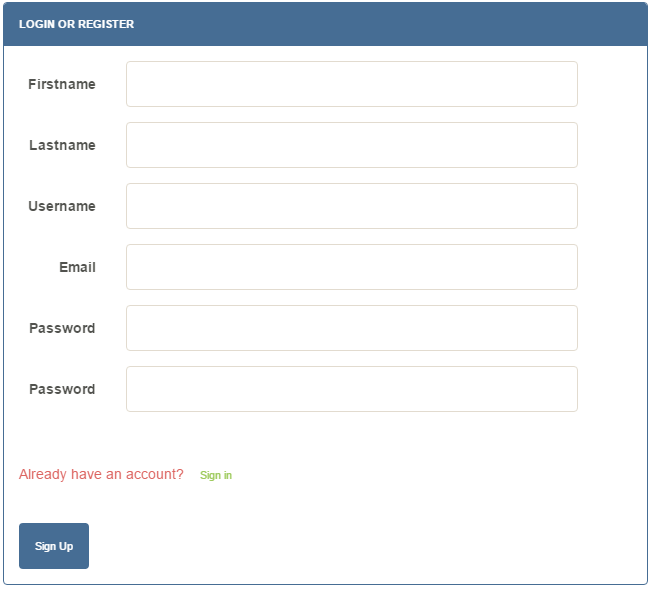


**Figure 4.2:** record stored in transfer.users table during application testing.

### 4.4 SCRENSHOTS OF THE WEBSITE DESIGN PAGES

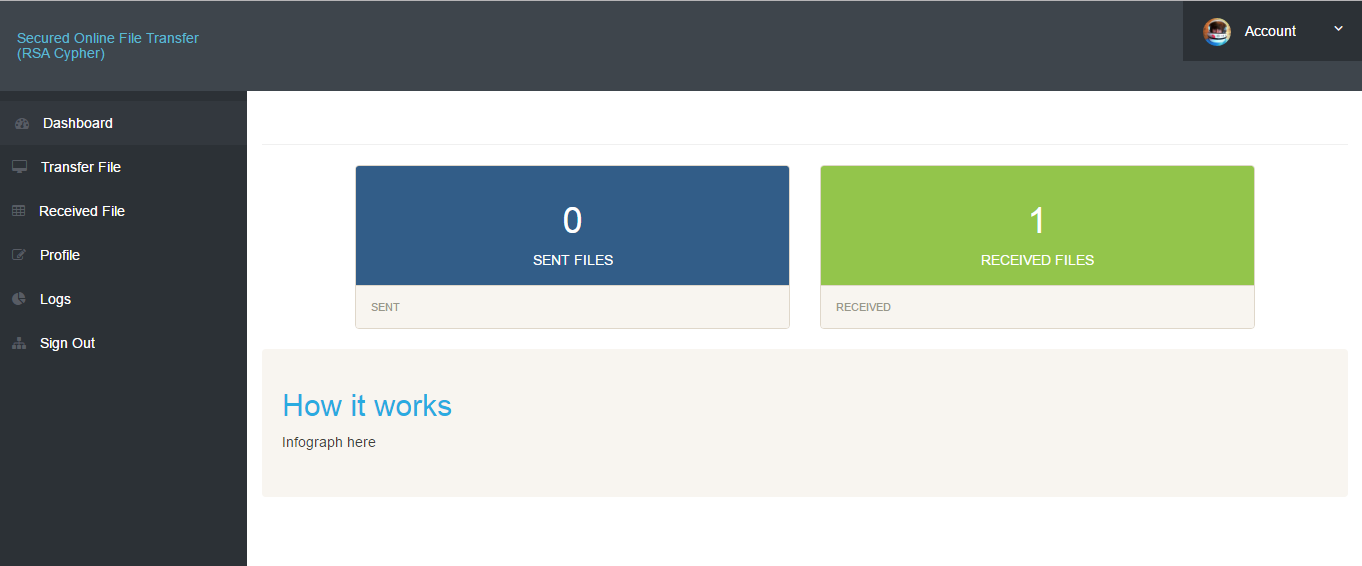
The landing page in figure 4.3 show the screenshost of where the user can sign up or login from which serve as the home page either through a referral link or directly.





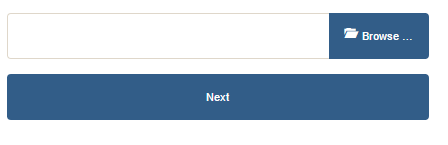
**Figure 4.3:** Website homepage and its constituent dashboards.

The figure 4.4 below shows the page that is first displayed after successful signup or login process. This page projects summary count of number of files being sent, and total number of file being received.

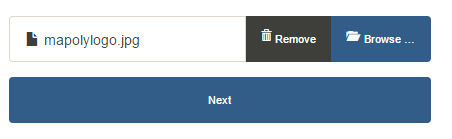


**Figure 4.4:** the website secured file transfer home dashboard.

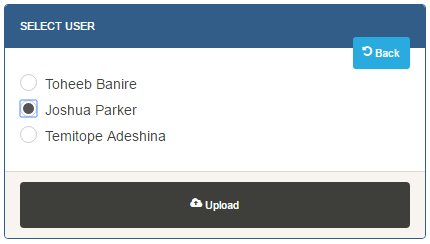
The figure 4.5 below project the procedures involve in sending a file to the user and all in one they make up the transfer dashboard.



Browse and select a file to transfer



Display outlook after a file is selected for transfer



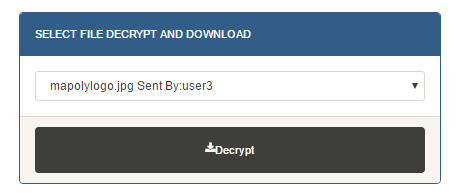
Select a user to transfer file to and the upload button performs the encryption and transfer action



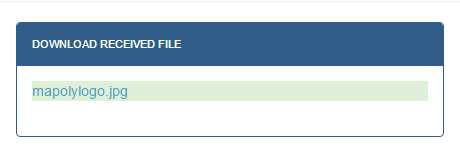
Display message after transfer is successful

**Figure 4.5:** screenshot of file transfer process dashboards.

The screenshot in figure 4.6 below expresses the reception and decryption processes since the application is designed for the purpose that logged in user can only access authorized file i.e. file being sent by the user or files being sent to the user.



Select a file to decrypt and (view or download) form list of received files and click decrypt buttion.



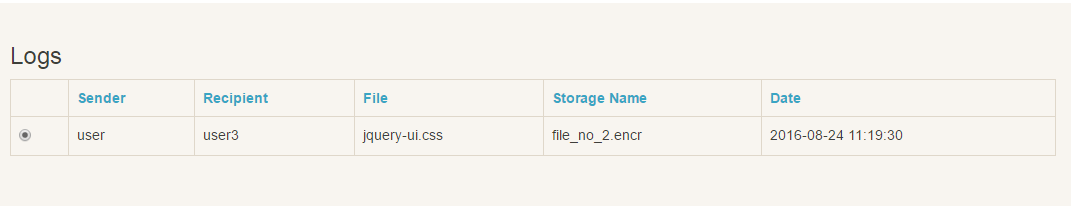
Display link of decrypted file to access and download



Display true file content of decrypted file on browser or as a downloaded file.

**Figure 4.6:** screenshot of file reception process dashboards.

The screenshot below in figure 4.7 shows the log of files being sent or received by a specific user as a table.



**Figure 4.7:** the website user’s file transfer log dashboard

# CHAPTER FIVE

## 5.0 RECOMMENDATIONS AND CONCLUSION

### 5.1 RECOMMENDATIONS

Secured file transfer system using RSA cipher, show case the possibility of saving communicating a file between user on a website platform in such a way that the platform owner or online intruders will find it extremely difficult or impossible to intercept a file that is being transferred on internet platform. For the fact that once a user goes live or online, his or her communication is open to interception by intruders due to lack of application of computer security mechanisms. It was a few years back in the beginning of the millennium teens i.e. 2013 that telegram chatting system first came up with the idea of enciphering of chats and messages before whatsapp also join its usage as at last year but The focus of their encryption is on message but not extended to file being communicated on the chatting platform which successively calls for the recommendation of this project in education the public application development to the introduction of enciphering and deciphering of files on a communication platform.

### 5.2 CONCLUSION

The project secured file transfer system using RSA cipher is achieved by putting varieties of resources together starting from the research breakdown of studying the list of most commonly and strongly used cryptographic algorithms for the security of files and data in transit over a communication channel and ways through which files can be transferred or communicated between parties with use of computer based driven resources. The file security platform is place on cryptography while RSA is proficiently chosen because it is recorded among the first ten strongest cipher that is being used in securing files and message communication as of today. The web platform is used as focus of design because it appears to be one of the information and communication technology platform vulnerable to information attack.

# REFERENCES

*Euclidean algorithm.* (2015, August 8). Retrieved from Wikipedia.com: https://en.wikipedia.org/wiki/Euclidean\_algorithm

*File sharing.* (2013, February 23). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/File\_sharing#History

*File Transfer Protocol.* (2015, July 15). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/File\_Transfer\_Protocol

Jun, X. H. (2007). *A Boosting Algorithm for Information Retrieval.* AdaRank.

Margaret, R. (2013). RSA popularity.

Richard, A., & Peter, K. (2015). *File Management Overview.* American Society of Media Photographer. Retrieved from http://www.dpbestflow.org/file-management/file-management-overview

*RSA algorithm (Rivest-Shamir-Adleman) .* (2006). Retrieved from Techtarget: http://searchsecurity.techtarget.com/definition/RSA

Wells, D. (1989). *Prime Number Hide-and-Seek: How the RSA Cipher Works.*

# APPENDIX

1. **Index.php**

<?php

session\_start();

if (isset($\_SESSION['username'])) {

header('location:send.php');

}

?>

<?php require 'header.php'; ?>

<div class="well-lg"></div>

<div class="col-md-6 col-md-offset-3">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title"><strong>Login Or Register</strong></h3>

</div>

<div class="panel-body">

<div class="loginform">

<form method="POST" id="login" class="form-horizontal">

<div class="form-group">

<label for="username" class="control-label col-xs-2">Username</label>

<div class="col-xs-9">

<input type="text" id="username" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="email" class="control-label col-xs-2">Password</label>

<div class="col-xs-9">

<input type="password" id="password" class="form-control" required>

</div>

</div>

<p id="hint" class="text-primary"></p>

<p class="text-danger">Don't have an account?<a id="to-reg" class="btn btn-link">Sign Up</a></p>

<button type="submit" class="btn btn-primary"><span class="glyphicon glyphicon-log-in"></span>&nbsp;&nbsp;Sign in</button>

</form>

</div>

<div class="regform">

<form method="POST" id="register" class="form-horizontal">

<div class="form-group">

<label for="fname" class="control-label col-xs-2">Firstname</label>

<div class="col-xs-9">

<input type="text" id="fname" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="lname" class="control-label col-xs-2">Lastname</label>

<div class="col-xs-9">

<input type="text" id="lname" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="email" class="control-label col-xs-2">Username</label>

<div class="col-xs-9">

<input type="text" id="user\_name" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="email" class="control-label col-xs-2">Email</label>

<div class="col-xs-9">

<input type="email" id="mail" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="email" class="control-label col-xs-2">Password</label>

<div class="col-xs-9">

<input type="password" id="paswd" class="form-control" required>

</div>

</div>

<div class="form-group">

<label for="email" class="control-label col-xs-2">Password</label>

<div class="col-xs-9">

<input type="password" id="repaswd" class="form-control" required>

</div>

</div>

<div class="clearfix"></div>

<p id="reg-hint" class="text-primary"></p>

<p class="text-danger">Already have an account?<a id="to-login" class="btn btn-link">Sign in</a></p>

<button type="submit" class="btn btn-primary">Sign Up</button>

</form>

</div>

</div>

</div>

</div>

<?php require 'footer.php'; ?>

</body>

</html>

1. **Profile.php**

<?php require '../core/config.php'; ?>

<?php require 'header.php'; ?>

<?php require 'dashheader.php'; ?>

<?php

session\_start();

$user = $\_SESSION['username'];

$opts['hn'] = DB\_HOST;

$opts['un'] = DB\_USER;

$opts['pw'] = DB\_PASS;

$opts['db'] = DB\_NAME;

$opts['tb'] = "users";

$opts['logtable'] = $log\_table;

// Name of field which is the unique key

$opts['key'] = 'id';

// Type of key field (int/real/string/date etc.)

$opts['key\_type'] = 'int';

// Sorting field(s)

$opts['sort\_field'] = array('id');

// Number of records to display on the screen

// Value of -1 lists all records in a table

$opts['inc'] = 15;

$opts['options'] = 'ACPVD';

//filter

$opts['filters'] = "username = '$user'";

// Number of lines to display on multiple selection filters

$opts['multiple'] = '4';

$opts['navigation'] = 'DB';

$opts['buttons']['L']['down'] = array('change');

//Custom buttons

$opts['buttons']['C']['down'] = array(

'<input type="hidden" name="PME\_sys\_savechange" value="Save" />',

'<button class="btn btn-primary btn-sm" name="Submit" id="Submit" onclick="return PME\_js\_form\_control(this.form);" type="submit">

Update</button>',

'<a href="" class="btn btn-warning btn-sm">Back</a>',

''

);

$opts['fdd']['id'] = array(

'name' => 'ID',

'select' => 'T',

'options' => 'AVCPDRH', // auto increment

'maxlen' => 11,

'default' => '0',

'sort' => true

);

$opts['fdd']['fname'] = array(

'name' => 'First Name',

'select' => 'T',

'maxlen' => 32,

'sort' => true

);

$opts['fdd']['lname'] = array(

'name' => 'Last Name',

'select' => 'T',

'maxlen' => 32,

'sort' => true

);

$opts['fdd']['email'] = array(

'name' => 'Email',

'select' => 'T',

'maxlen' => 10,

'sort' => true

);

$opts['fdd']['username'] = array(

'name' => 'Username',

'input' => 'R',

'select' => 'T',

'maxlen' => 32,

'sort' => true

);

$opts['fdd']['password'] = array(

'name' => 'Password',

'select' => 'T',

'maxlen' => 255,

'sort' => true

);

echo '<form role="form" method="post" class="edit\_form" style="padding:20px;">';

echo '<h3>Profile</h3>';

require\_once "../core/phpMyEdit.class.php";

new phpMyEdit($opts);

$the\_table = $opts['tb'];

echo '<div class="form-group"></div></form>';

?>

<?php require 'dashfooter.php'; ?>

1. **File\_Processor.php**

<?php

session\_start();

// check if file is present in variable $\_REQUEST['file'] and

// user selected in variable $\_REQUEST['recipient']

// if not return to transfer.php

require\_once \_\_DIR\_\_ . '/../phpseclib-php5-master/phpseclib/Crypt/RSA.php';

require\_once \_\_DIR\_\_ . '/../core/Db\_connect.php';

$db = new Db\_connect();

$encrypted\_path = "encrypted\_files/";

$decrypted\_path = "decrypted\_files/";

if (isset($\_FILES['file'], $\_REQUEST['recipient'])) {

require 'header.php';

require 'dashheader.php';

$source\_file = basename($\_FILES['file']["name"]);

$newName= $db->save\_sent\_file($\_SESSION['username'], $source\_file, $\_REQUEST['recipient']);

// echo "$newName $encrypted\_path";

if ($newName!==0) {

$byteFile = convert\_to\_byte($\_FILES['file']);

$result = file\_put\_contents("$encrypted\_path$newName",$byteFile);

if ($result==FALSE) {

echo '<div class="alert alert-warning">

<a class="close" data-dismiss="alert">x</a><h4>Error</h4>

Error Encrypting and Transferring File!!!

</div>';

}

echo '<div class="alert alert-success">

<a class="close" data-dismiss="alert">x</a><h4>Success</h4>

Encryption and Transer is Successful!!!</div>';

}

require 'dashfooter.php';

require 'footer.php';

} else {

header("location:transfer.php?x=0");

}

function convert\_to\_byte($param) {

return file\_get\_contents($param["tmp\_name"]);

}

function save\_file\_to\_dir($sourceFile, $target) {

if (file\_exists($target)) {

return false;

}

return file\_put\_contents($target, $sourceFile);

}

1. **Transfer.php**

<?php require 'header.php'; ?>

<?php require 'dashheader.php'; ?>

<?php

require\_once \_\_DIR\_\_ . '/../phpseclib-php5-master/phpseclib/Crypt/RSA.php';

require '../core/Db\_connect.php';

$db = new Db\_connect();

$users = $db->get\_users();

?>

<form action="file\_processor.php" method="post" enctype="multipart/form-data">

<div id="transfer">

<div class="col-md-5 col-md-offset-3" id="file">

<div class="form-group">

<input id="input-43" name="file" class="t" type="file">

</div>

<?php

if (isset($\_REQUEST['x'])) {

?>

<div class="alert alert-warning">

<a class="close" data-dismiss="alert">x</a><h4>Error</h4>

Error in file transfer a receiver is not selected<br/>

You need to select a receiver!!!

</div>

<?php }

?>

<button type="button" class="btn btn-block btn-primary" id="next">Next</button>

</div>

<div id="send-to" class="col-md-5 col-md-offset-3">

<div class="panel panel-primary">

<div class="panel-heading">

<h4 class="panel-title">Select User</h4>

<button type="button" id="back" class="btn btn-info btn-sm pull-right"><i class="fa fa-undo"></i> Back</button>

</div>

<div class="panel-body">

<?php foreach ($users as $key => $user): ?>

<div class="radio">

<input type="radio" name="recipient" id="radio<?php echo $key; ?>" value="<?php echo $user['username']; ?>" >

<label for="radio<?php echo $key; ?>">

<?php echo $user['fname'] . " " . $user['lname']; ?>

</label>

</div>

<?php endforeach; ?>

</div>

<div class="panel-footer">

<button type="submit" class="btn btn-default btn-block"><i class="fa fa-cloud-upload" aria-hidden="true"></i> Upload</button>

</div>

</div>

</div>

</div>

</form>

<?php require 'dashfooter.php'; ?>

<?php require 'footer.php'; ?>

1. **Receive.php**

<?php

require '../core/config.php';

require 'header.php';

require 'dashheader.php';

session\_start();

$user = $\_SESSION['username'];

require 'received\_form.php';

$opts['hn'] = DB\_HOST;

$opts['un'] = DB\_USER;

$opts['pw'] = DB\_PASS;

$opts['db'] = DB\_NAME;

$opts['tb'] = "file\_transfer";

$opts['logtable'] = $log\_table;

// Name of field which is the unique key

$opts['key'] = 'id';

$opts['key\_type'] = 'int';

// Sorting field(s)

$opts['sort\_field'] = array('id');

$opts['inc'] = 15;

$opts['options'] = 'ACPVD';

//filter

$opts['filters'] = "receiver = '$user'";

$opts['multiple'] = '4';

$opts['navigation'] = 'DB';

$opts['buttons']['L']['down'] = array('');

$opts['fdd']['id'] = array(

'name' => 'ID',

'select' => 'T',

'options' => 'AVCPDRH', // auto increment

'maxlen' => 11,

'default' => '0',

'sort' => true

);

$opts['fdd']['sender'] = array(

'name' => 'Sender',

'select' => 'T',

'maxlen' => 32,

'help|AC' => 'The sender of the file',

'sort' => true

);

$opts['fdd']['receiver'] = array(

'name' => 'Recipient',

'select' => 'T',

'maxlen' => 32,

'sort' => true

);

$opts['fdd']['file\_addr'] = array(

'name' => 'File',

'select' => 'T',

'maxlen' => 200,

'sort' => true

);

$opts['fdd']['date'] = array(

'name' => 'Date',

'input' => 'R',

'select' => 'T',

'maxlen' => 32,

'sort' => true

);

echo '<form role="form" method="post" class="edit\_form" style="padding:20px;">';

echo '<h3>Received Files</h3>';

require\_once "../core/phpMyEdit.class.php";

new phpMyEdit($opts);

$the\_table = $opts['tb'];

echo '<div class="form-group"></div></form>';

?>

<?php require 'dashfooter.php'; ?>

1. **Send.php**

<?php

session\_start();

if (!isset($\_SESSION['username'])) {

header('location:index.php');

}

require\_once \_\_DIR\_\_ . '/../core/Db\_connect.php';

$chat = new Db\_connect();

$users = $chat->get\_users();

$counts = $chat->get\_sender();

$recev = $chat->get\_receiver();

foreach ($counts as $key=> $count){

}

foreach ($recev as $key=> $rece){

}

?>

<?php require 'header.php'; ?>

<?php require 'dashheader.php'; ?>

<div class="row">

<div class="col-md-5 col-md-offset-1">

<div class="panel panel-default">

<div class="panel-body bk-primary text-light">

<div class="stat-panel text-center">

<div class="stat-panel-number h1 "><?php echo $count['sid']; ?></div>

<div class="stat-panel-title text-uppercase">Sent Files</div>

</div>

</div>

<a href="logs.php?p=send" class="panel-footer block-anchor" >sent</a>

</div>

</div>

<div class="col-md-5">

<div class="panel panel-default">

<div class="panel-body bk-success text-light">

<div class="stat-panel text-center">

<div class="stat-panel-number h1 "><?php echo $rece['sid']; ?></div>

<div class="stat-panel-title text-uppercase">Received Files</div>

</div>

</div>

<a href="logs.php?p=received" class="panel-footer block-anchor" >Received</a>

</div>

</div>

</div>

<div class="row">

<div class="col-md-12">

<div class="well">

<h2 class="text-info">How it works</h2>

<p>Infograph here</p>

</div>

</div>

</div>

<?php require 'dashfooter.php'; ?>

<!-- Loading Scripts -->

<?php require 'footer.php'; ?>

</body>

</html>

1. **Receive\_Form.php**

<?php

require '../core/Db\_connect.php';

$db = new Db\_connect();

$encrypted\_path = "encrypted\_files/";

$decrypted\_path = "decrypted\_files/";

if (!file\_exists("$decrypted\_path$user/")) {

mkdir("$decrypted\_path$user/");

}

$thf['file\_addr']=$\_SESSION['curr\_file'];

if (isset($\_REQUEST['decrypt'])) {

$flid = explode("\_", $\_REQUEST['received\_files'])[0];

$thefile = $db->get\_results("SELECT \* FROM `file\_transfer` WHERE `receiver` = '$user' AND `id`='$flid';");

foreach ($thefile as $thf) {

};

// delete();

$newname = explode( ".",basename($encrypted\_path . $thf['temp\_name']))[0] . ".txt";

rename($encrypted\_path . $thf['temp\_name'],$encrypted\_path .$newname);

$byteFile1 = file\_get\_contents($encrypted\_path . $newname);

// echo $byteFile1;

$result = file\_put\_contents("$decrypted\_path$user/".$thf['file\_addr'] , $byteFile1);

$\_SESSION['curr\_file']=$thf['file\_addr'];

unlink($encrypted\_path.$newname);

$db->query\_db("DELETE FROM `file\_transfer` WHERE `receiver`='$user' AND `id`='$flid';");

?>

<div class="row">

<div class="col-md-5 col-md-offset-3">

<div class="panel panel-primary">

<div class="panel-heading">

<h4 class="panel-title">Download Received File</h4>

</div>

<div class="panel-body">

<ul class="list-group list-group-item-success">

<?php echo '<li><a href="' . $decrypted\_path . $user . "/" . $thf['file\_addr'] . '?h=' . $decrypted\_path . $user . "/" . $thf['file\_addr']. '">' . $thf['file\_addr'] . '</a></li>'; ?>

</ul>

</div>

</div>

</div>

</div>

<?php

}

if (isset($\_REQUEST['h'])) {

$\_SESSION['curr\_file']='';

// unlink($\_REQUEST['h']);

echo '<div class="alert alert-success">

<a class="close" data-dismiss="alert">x</a><h4>Success</h4>

Decrypted File is Successfully Downloaded!!!</div>';

}

$files = $db->get\_received\_files($user);

?>

<form action="received.php" method="post">

<div id="receiveds">

<div class="col-md-5 col-md-offset-3">

<div class="panel panel-primary">

<div class="panel-heading">

<h4 class="panel-title">Select File Decrypt and Download</h4>

</div>

<div class="panel-body">

<select class="form-control input-sm" name="received\_files">

<?php foreach ($files as $fl): ?>

<option id="radio<?php echo $fl['id']; ?>" value="<?php echo $fl['id'] . "\_" . $fl['temp\_name']; ?>" >

<?php echo $fl['file\_addr'] . " Sent By:" . $fl['sender']; ?>

</option>

<?php endforeach; ?>

</select>

</div>

<div class="panel-footer">

<button type="submit" class="btn btn-default btn-block" name="decrypt" value="decrypt\_me"><i class="fa fa-download" aria-hidden="true"></i>Decrypt</button>

</div>

</div>

</div>

</div>

</form>

1. **RSA.php**

<?php

/\*\*

\* Include Crypt\_Random

\*/

if (!function\_exists('crypt\_random\_string')) {

require\_once('Random.php');

}

/\*\*

\* Include Crypt\_Hash

\*/

if (!class\_exists('Crypt\_Hash')) {

require\_once('Hash.php');

}

/\*\*#@+

\* @access public

\* @see Crypt\_RSA::encrypt()

\* @see Crypt\_RSA::decrypt()

\*/

define('CRYPT\_RSA\_ENCRYPTION\_OAEP', 1);

define('CRYPT\_RSA\_ENCRYPTION\_PKCS1', 2);

define('CRYPT\_RSA\_SIGNATURE\_PSS', 1);

define('CRYPT\_RSA\_SIGNATURE\_PKCS1', 2);

define('CRYPT\_RSA\_ASN1\_INTEGER', 2);

define('CRYPT\_RSA\_ASN1\_BITSTRING', 3);

define('CRYPT\_RSA\_ASN1\_SEQUENCE', 48);

define('CRYPT\_RSA\_MODE\_INTERNAL', 1);

define('CRYPT\_RSA\_MODE\_OPENSSL', 2);

define('CRYPT\_RSA\_OPENSSL\_CONFIG', dirname(\_\_FILE\_\_) . '/../openssl.cnf');

define('CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1', 0);

/\*\*

\* PuTTY formatted private key

\*/

define('CRYPT\_RSA\_PRIVATE\_FORMAT\_PUTTY', 1);

define('CRYPT\_RSA\_PRIVATE\_FORMAT\_XML', 2);

define('CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW', 3);

define('CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1\_RAW', 4);

define('CRYPT\_RSA\_PUBLIC\_FORMAT\_XML', 5);

define('CRYPT\_RSA\_PUBLIC\_FORMAT\_OPENSSH', 6);

define('CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1', 7);

class MultiPrimeNotSupportedException extends Exception {}

class FormatNotSupportedException extends Exception {}

class KeyNotLoadedException extends Exception {}

class EncryptionException extends Exception {}

class DecryptionException extends Exception {}

class SigningException extends Exception {}

class VerificationException extends Exception {}

class EncodingException extends Exception {}

class BadPasswordException extends Exception {}

class Crypt\_RSA {

var $privateKeyFormat = CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1;

var $publicKeyFormat = CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1;

var $encryptionMode = CRYPT\_RSA\_ENCRYPTION\_OAEP;

var $signatureMode = CRYPT\_RSA\_SIGNATURE\_PSS;

var $publicExponent = false;

/\*\*

\* Password

\*

\* @var String

\* @access private

\*/

var $password = false;

var $components = array();

var $comment = 'phpseclib-generated-key';

function Crypt\_RSA()

{

if (!class\_exists('Math\_BigInteger')) {

require\_once('Math/BigInteger.php');

}

$this->configFile = CRYPT\_RSA\_OPENSSL\_CONFIG;

if ( !defined('CRYPT\_RSA\_MODE') ) {

switch (true) {

case extension\_loaded('openssl') && version\_compare(PHP\_VERSION, '4.2.0', '>=') && file\_exists($this->configFile):

define('CRYPT\_RSA\_MODE', CRYPT\_RSA\_MODE\_OPENSSL);

break;

default:

define('CRYPT\_RSA\_MODE', CRYPT\_RSA\_MODE\_INTERNAL);

}

}

$this->zero = new Math\_BigInteger();

$this->one = new Math\_BigInteger(1);

$this->hash = new Crypt\_Hash('sha1');

$this->hLen = $this->hash->getLength();

$this->hashName = 'sha1';

$this->mgfHash = new Crypt\_Hash('sha1');

$this->mgfHLen = $this->mgfHash->getLength();

}

function createKey($bits = 1024, $timeout = false, $partial = array())

{

if (!defined('CRYPT\_RSA\_EXPONENT')) {

// http://en.wikipedia.org/wiki/65537\_%28number%29

define('CRYPT\_RSA\_EXPONENT', '65537');

}

// per <http://cseweb.ucsd.edu/~hovav/dist/survey.pdf#page=5>, this number ought not result in primes smaller

// than 256 bits. as a consequence if the key you're trying to create is 1024 bits and you've set CRYPT\_RSA\_SMALLEST\_PRIME

// to 384 bits then you're going to get a 384 bit prime and a 640 bit prime (384 + 1024 % 384). at least if

// CRYPT\_RSA\_MODE is set to CRYPT\_RSA\_MODE\_INTERNAL. if CRYPT\_RSA\_MODE is set to CRYPT\_RSA\_MODE\_OPENSSL then

// CRYPT\_RSA\_SMALLEST\_PRIME is ignored (ie. multi-prime RSA support is more intended as a way to speed up RSA key

// generation when there's a chance neither gmp nor OpenSSL are installed)

if (!defined('CRYPT\_RSA\_SMALLEST\_PRIME')) {

define('CRYPT\_RSA\_SMALLEST\_PRIME', 4096);

}

// OpenSSL uses 65537 as the exponent and requires RSA keys be 384 bits minimum

if ( CRYPT\_RSA\_MODE == CRYPT\_RSA\_MODE\_OPENSSL && $bits >= 384 && CRYPT\_RSA\_EXPONENT == 65537) {

$config = array();

if (isset($this->configFile)) {

$config['config'] = $this->configFile;

}

$rsa = openssl\_pkey\_new(array('private\_key\_bits' => $bits) + $config);

openssl\_pkey\_export($rsa, $privatekey, NULL, $config);

$publickey = openssl\_pkey\_get\_details($rsa);

$publickey = $publickey['key'];

$privatekey = call\_user\_func\_array(array($this, '\_convertPrivateKey'), array\_values($this->\_parseKey($privatekey, CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1)));

$publickey = call\_user\_func\_array(array($this, '\_convertPublicKey'), array\_values($this->\_parseKey($publickey, CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1)));

// clear the buffer of error strings stemming from a minimalistic openssl.cnf

while (openssl\_error\_string() !== false);

return array(

'privatekey' => $privatekey,

'publickey' => $publickey,

'partialkey' => false

);

}

static $e;

if (!isset($e)) {

$e = new Math\_BigInteger(CRYPT\_RSA\_EXPONENT);

}

extract($this->\_generateMinMax($bits));

$absoluteMin = $min;

$temp = $bits >> 1; // divide by two to see how many bits P and Q would be

if ($temp > CRYPT\_RSA\_SMALLEST\_PRIME) {

$num\_primes = floor($bits / CRYPT\_RSA\_SMALLEST\_PRIME);

$temp = CRYPT\_RSA\_SMALLEST\_PRIME;

} else {

$num\_primes = 2;

}

extract($this->\_generateMinMax($temp + $bits % $temp));

$finalMax = $max;

extract($this->\_generateMinMax($temp));

$generator = new Math\_BigInteger();

$n = $this->one->copy();

if (!empty($partial)) {

extract(unserialize($partial));

} else {

$exponents = $coefficients = $primes = array();

$lcm = array(

'top' => $this->one->copy(),

'bottom' => false

);

}

$start = time();

$i0 = count($primes) + 1;

do {

for ($i = $i0; $i <= $num\_primes; $i++) {

if ($timeout !== false) {

$timeout-= time() - $start;

$start = time();

if ($timeout <= 0) {

return array(

'privatekey' => '',

'publickey' => '',

'partialkey' => serialize(array(

'primes' => $primes,

'coefficients' => $coefficients,

'lcm' => $lcm,

'exponents' => $exponents

))

);

}

}

if ($i == $num\_primes) {

list($min, $temp) = $absoluteMin->divide($n);

if (!$temp->equals($this->zero)) {

$min = $min->add($this->one); // ie. ceil()

}

$primes[$i] = $generator->randomPrime($min, $finalMax, $timeout);

} else {

$primes[$i] = $generator->randomPrime($min, $max, $timeout);

}

if ($primes[$i] === false) { // if we've reached the timeout

if (count($primes) > 1) {

$partialkey = '';

} else {

array\_pop($primes);

$partialkey = serialize(array(

'primes' => $primes,

'coefficients' => $coefficients,

'lcm' => $lcm,

'exponents' => $exponents

));

}

return array(

'privatekey' => '',

'publickey' => '',

'partialkey' => $partialkey

);

}

// the first coefficient is calculated differently from the rest

// ie. instead of being $primes[1]->modInverse($primes[2]), it's $primes[2]->modInverse($primes[1])

if ($i > 2) {

$coefficients[$i] = $n->modInverse($primes[$i]);

}

$n = $n->multiply($primes[$i]);

$temp = $primes[$i]->subtract($this->one);

// textbook RSA implementations use Euler's totient function instead of the least common multiple.

// see http://en.wikipedia.org/wiki/Euler%27s\_totient\_function

$lcm['top'] = $lcm['top']->multiply($temp);

$lcm['bottom'] = $lcm['bottom'] === false ? $temp : $lcm['bottom']->gcd($temp);

$exponents[$i] = $e->modInverse($temp);

}

list($lcm) = $lcm['top']->divide($lcm['bottom']);

$gcd = $lcm->gcd($e);

$i0 = 1;

} while (!$gcd->equals($this->one));

$d = $e->modInverse($lcm);

$coefficients[2] = $primes[2]->modInverse($primes[1]);

return array(

'privatekey' => $this->\_convertPrivateKey($n, $e, $d, $primes, $exponents, $coefficients),

'publickey' => $this->\_convertPublicKey($n, $e),

'partialkey' => false

);

}

function \_convertPrivateKey($n, $e, $d, $primes, $exponents, $coefficients)

{

$num\_primes = count($primes);

$raw = array(

'version' => $num\_primes == 2 ? chr(0) : chr(1), // two-prime vs. multi

'modulus' => $n->toBytes(true),

'publicExponent' => $e->toBytes(true),

'privateExponent' => $d->toBytes(true),

'prime1' => $primes[1]->toBytes(true),

'prime2' => $primes[2]->toBytes(true),

'exponent1' => $exponents[1]->toBytes(true),

'exponent2' => $exponents[2]->toBytes(true),

'coefficient' => $coefficients[2]->toBytes(true)

);

// if the format in question does not support multi-prime rsa and multi-prime rsa was used,

// call \_convertPublicKey() instead.

switch ($this->privateKeyFormat) {

case CRYPT\_RSA\_PRIVATE\_FORMAT\_XML:

if ($num\_primes != 2) {

throw new MultiPrimeNotSupportedException('Multi-prime RSA is not supported by XML keys');

}

return "<RSAKeyValue>\r\n" .

' <Modulus>' . base64\_encode($raw['modulus']) . "</Modulus>\r\n" .

' <Exponent>' . base64\_encode($raw['publicExponent']) . "</Exponent>\r\n" .

' <P>' . base64\_encode($raw['prime1']) . "</P>\r\n" .

' <Q>' . base64\_encode($raw['prime2']) . "</Q>\r\n" .

' <DP>' . base64\_encode($raw['exponent1']) . "</DP>\r\n" .

' <DQ>' . base64\_encode($raw['exponent2']) . "</DQ>\r\n" .

' <InverseQ>' . base64\_encode($raw['coefficient']) . "</InverseQ>\r\n" .

' <D>' . base64\_encode($raw['privateExponent']) . "</D>\r\n" .

'</RSAKeyValue>';

break;

case CRYPT\_RSA\_PRIVATE\_FORMAT\_PUTTY:

if ($num\_primes != 2) {

throw new MultiPrimeNotSupportedException('Multi-prime RSA is not supported by PuTTY keys');

}

$key = "PuTTY-User-Key-File-2: ssh-rsa\r\nEncryption: ";

$encryption = (!empty($this->password) || is\_string($this->password)) ? 'aes256-cbc' : 'none';

$key.= $encryption;

$key.= "\r\nComment: " . $this->comment . "\r\n";

$public = pack('Na\*Na\*Na\*',

strlen('ssh-rsa'), 'ssh-rsa', strlen($raw['publicExponent']), $raw['publicExponent'], strlen($raw['modulus']), $raw['modulus']

);

$source = pack('Na\*Na\*Na\*Na\*',

strlen('ssh-rsa'), 'ssh-rsa', strlen($encryption), $encryption,

strlen($this->comment), $this->comment, strlen($public), $public

);

$public = base64\_encode($public);

$key.= "Public-Lines: " . ((strlen($public) + 32) >> 6) . "\r\n";

$key.= chunk\_split($public, 64);

$private = pack('Na\*Na\*Na\*Na\*',

strlen($raw['privateExponent']), $raw['privateExponent'], strlen($raw['prime1']), $raw['prime1'],

strlen($raw['prime2']), $raw['prime2'], strlen($raw['coefficient']), $raw['coefficient']

);

if (empty($this->password) && !is\_string($this->password)) {

$source.= pack('Na\*', strlen($private), $private);

$hashkey = 'putty-private-key-file-mac-key';

} else {

$private.= crypt\_random\_string(16 - (strlen($private) & 15));

$source.= pack('Na\*', strlen($private), $private);

if (!class\_exists('Crypt\_AES')) {

require\_once('Crypt/AES.php');

}

$sequence = 0;

$symkey = '';

while (strlen($symkey) < 32) {

$temp = pack('Na\*', $sequence++, $this->password);

$symkey.= pack('H\*', sha1($temp));

}

$symkey = substr($symkey, 0, 32);

$crypto = new Crypt\_AES();

$crypto->setKey($symkey);

$crypto->disablePadding();

$private = $crypto->encrypt($private);

$hashkey = 'putty-private-key-file-mac-key' . $this->password;

}

$private = base64\_encode($private);

$key.= 'Private-Lines: ' . ((strlen($private) + 32) >> 6) . "\r\n";

$key.= chunk\_split($private, 64);

if (!class\_exists('Crypt\_Hash')) {

require\_once('Crypt/Hash.php');

}

$hash = new Crypt\_Hash('sha1');

$hash->setKey(pack('H\*', sha1($hashkey)));

$key.= 'Private-MAC: ' . bin2hex($hash->hash($source)) . "\r\n";

return $key;

default: // eg. CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1

$components = array();

foreach ($raw as $name => $value) {

$components[$name] = pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($value)), $value);

}

$RSAPrivateKey = implode('', $components);

if ($num\_primes > 2) {

$OtherPrimeInfos = '';

for ($i = 3; $i <= $num\_primes; $i++) {

// OtherPrimeInfos ::= SEQUENCE SIZE(1..MAX) OF OtherPrimeInfo

//

// OtherPrimeInfo ::= SEQUENCE {

// prime INTEGER, -- ri

// exponent INTEGER, -- di

// coefficient INTEGER -- ti

// }

$OtherPrimeInfo = pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($primes[$i]->toBytes(true))), $primes[$i]->toBytes(true));

$OtherPrimeInfo.= pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($exponents[$i]->toBytes(true))), $exponents[$i]->toBytes(true));

$OtherPrimeInfo.= pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($coefficients[$i]->toBytes(true))), $coefficients[$i]->toBytes(true));

$OtherPrimeInfos.= pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_SEQUENCE, $this->\_encodeLength(strlen($OtherPrimeInfo)), $OtherPrimeInfo);

}

$RSAPrivateKey.= pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_SEQUENCE, $this->\_encodeLength(strlen($OtherPrimeInfos)), $OtherPrimeInfos);

}

$RSAPrivateKey = pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_SEQUENCE, $this->\_encodeLength(strlen($RSAPrivateKey)), $RSAPrivateKey);

if (!empty($this->password) || is\_string($this->password)) {

$iv = crypt\_random\_string(8);

$symkey = pack('H\*', md5($this->password . $iv)); // symkey is short for symmetric key

$symkey.= substr(pack('H\*', md5($symkey . $this->password . $iv)), 0, 8);

if (!class\_exists('Crypt\_TripleDES')) {

require\_once('Crypt/TripleDES.php');

}

$des = new Crypt\_TripleDES();

$des->setKey($symkey);

$des->setIV($iv);

$iv = strtoupper(bin2hex($iv));

$RSAPrivateKey = "-----BEGIN RSA PRIVATE KEY-----\r\n" .

"Proc-Type: 4,ENCRYPTED\r\n" .

"DEK-Info: DES-EDE3-CBC,$iv\r\n" .

"\r\n" .

chunk\_split(base64\_encode($des->encrypt($RSAPrivateKey)), 64) .

'-----END RSA PRIVATE KEY-----';

} else {

$RSAPrivateKey = "-----BEGIN RSA PRIVATE KEY-----\r\n" .

chunk\_split(base64\_encode($RSAPrivateKey), 64) .

'-----END RSA PRIVATE KEY-----';

}

return $RSAPrivateKey;

}

}

/\*\*

\* Convert a public key to the appropriate format

\*

\* @access private

\* @see setPublicKeyFormat()

\* @param String $RSAPrivateKey

\* @return String

\*/

function \_convertPublicKey($n, $e)

{

$modulus = $n->toBytes(true);

$publicExponent = $e->toBytes(true);

switch ($this->publicKeyFormat) {

case CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW:

return array('e' => $e->copy(), 'n' => $n->copy());

case CRYPT\_RSA\_PUBLIC\_FORMAT\_XML:

return "<RSAKeyValue>\r\n" .

' <Modulus>' . base64\_encode($modulus) . "</Modulus>\r\n" .

' <Exponent>' . base64\_encode($publicExponent) . "</Exponent>\r\n" .

'</RSAKeyValue>';

break;

case CRYPT\_RSA\_PUBLIC\_FORMAT\_OPENSSH:

// from <http://tools.ietf.org/html/rfc4253#page-15>:

// string "ssh-rsa"

// mpint e

// mpint n

$RSAPublicKey = pack('Na\*Na\*Na\*', strlen('ssh-rsa'), 'ssh-rsa', strlen($publicExponent), $publicExponent, strlen($modulus), $modulus);

$RSAPublicKey = 'ssh-rsa ' . base64\_encode($RSAPublicKey) . ' ' . $this->comment;

return $RSAPublicKey;

default: // eg. CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1\_RAW or CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1

// from <http://tools.ietf.org/html/rfc3447#appendix-A.1.1>:

// RSAPublicKey ::= SEQUENCE {

// modulus INTEGER, -- n

// publicExponent INTEGER -- e

// }

$components = array(

'modulus' => pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($modulus)), $modulus),

'publicExponent' => pack('Ca\*a\*', CRYPT\_RSA\_ASN1\_INTEGER, $this->\_encodeLength(strlen($publicExponent)), $publicExponent)

);

$RSAPublicKey = pack('Ca\*a\*a\*',

CRYPT\_RSA\_ASN1\_SEQUENCE, $this->\_encodeLength(strlen($components['modulus']) + strlen($components['publicExponent'])),

$components['modulus'], $components['publicExponent']

);

if ($this->publicKeyFormat == CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1) {

// sequence(oid(1.2.840.113549.1.1.1), null)) = rsaEncryption.

$rsaOID = pack('H\*', '300d06092a864886f70d0101010500'); // hex version of MA0GCSqGSIb3DQEBAQUA

$RSAPublicKey = chr(0) . $RSAPublicKey;

$RSAPublicKey = chr(3) . $this->\_encodeLength(strlen($RSAPublicKey)) . $RSAPublicKey;

$RSAPublicKey = pack('Ca\*a\*',

CRYPT\_RSA\_ASN1\_SEQUENCE, $this->\_encodeLength(strlen($rsaOID . $RSAPublicKey)), $rsaOID . $RSAPublicKey

);

}

$RSAPublicKey = "-----BEGIN PUBLIC KEY-----\r\n" .

chunk\_split(base64\_encode($RSAPublicKey), 64) .

'-----END PUBLIC KEY-----';

return $RSAPublicKey;

}

}

/\*\*

\* Break a public or private key down into its constituant components

\*

\* @access private

\* @see \_convertPublicKey()

\* @see \_convertPrivateKey()

\* @param String $key

\* @param Integer $type

\* @return Array

\*/

function \_parseKey($key, $type)

{

if ($type != CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW && !is\_string($key)) {

throw new FormatNotSupportedException('Unable to parse unsupported key format');

}

switch ($type) {

case CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW:

if (!is\_array($key)) {

throw new FormatNotSupportedException('Raw: Expected array type');

}

$components = array();

switch (true) {

case isset($key['e']):

$components['publicExponent'] = $key['e']->copy();

break;

case isset($key['exponent']):

$components['publicExponent'] = $key['exponent']->copy();

break;

case isset($key['publicExponent']):

$components['publicExponent'] = $key['publicExponent']->copy();

break;

case isset($key[0]):

$components['publicExponent'] = $key[0]->copy();

}

switch (true) {

case isset($key['n']):

$components['modulus'] = $key['n']->copy();

break;

case isset($key['modulo']):

$components['modulus'] = $key['modulo']->copy();

break;

case isset($key['modulus']):

$components['modulus'] = $key['modulus']->copy();

break;

case isset($key[1]):

$components['modulus'] = $key[1]->copy();

}

return isset($components['modulus']) && isset($components['publicExponent']) ? $components : false;

case CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1:

case CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1:

/\* Although PKCS#1 proposes a format that public and private keys can use, encrypting them is

"outside the scope" of PKCS#1. PKCS#1 then refers you to PKCS#12 and PKCS#15 if you're wanting to

protect private keys, however, that's not what OpenSSL\* does. OpenSSL protects private keys by adding

two new "fields" to the key - DEK-Info and Proc-Type. These fields are discussed here:

http://tools.ietf.org/html/rfc1421#section-4.6.1.1

http://tools.ietf.org/html/rfc1421#section-4.6.1.3

DES-EDE3-CBC as an algorithm, however, is not discussed anywhere, near as I can tell.

DES-CBC and DES-EDE are discussed in RFC1423, however, DES-EDE3-CBC isn't, nor is its key derivation

function. As is, the definitive authority on this encoding scheme isn't the IETF but rather OpenSSL's

own implementation. ie. the implementation \*is\* the standard and any bugs that may exist in that

implementation are part of the standard, as well.

\* OpenSSL is the de facto standard. It's utilized by OpenSSH and other projects \*/

$is\_encrypted = preg\_match('#DEK-Info: (.+),(.+)#', $key, $matches);

if ($is\_encrypted) {

$iv = pack('H\*', trim($matches[2]));

$symkey = pack('H\*', md5($this->password . substr($iv, 0, 8))); // symkey is short for symmetric key

$symkey.= pack('H\*', md5($symkey . $this->password . substr($iv, 0, 8)));

$ciphertext = preg\_replace('#.+(\r|\n|\r\n)\1|[\r\n]|-.+-| #s', '', $key);

$ciphertext = preg\_match('#^[a-zA-Z\d/+]\*={0,2}$#', $ciphertext) ? base64\_decode($ciphertext) : false;

if ($ciphertext === false) {

$ciphertext = $key;

}

switch ($matches[1]) {

case 'AES-256-CBC':

if (!class\_exists('Crypt\_AES')) {

require\_once('Crypt/AES.php');

}

$crypto = new Crypt\_AES();

break;

case 'AES-128-CBC':

if (!class\_exists('Crypt\_AES')) {

require\_once('Crypt/AES.php');

}

$symkey = substr($symkey, 0, 16);

$crypto = new Crypt\_AES();

break;

case 'DES-EDE3-CFB':

if (!class\_exists('Crypt\_TripleDES')) {

require\_once('Crypt/TripleDES.php');

}

$crypto = new Crypt\_TripleDES(CRYPT\_DES\_MODE\_CFB);

break;

case 'DES-EDE3-CBC':

if (!class\_exists('Crypt\_TripleDES')) {

require\_once('Crypt/TripleDES.php');

}

$symkey = substr($symkey, 0, 24);

$crypto = new Crypt\_TripleDES();

break;

case 'DES-CBC':

if (!class\_exists('Crypt\_DES')) {

require\_once('Crypt/DES.php');

}

$crypto = new Crypt\_DES();

break;

default:

throw new FormatNotSupportedException("PKCS1: $matches[1] is an unsupported cipher");

}

$crypto->setKey($symkey);

$crypto->setIV($iv);

try {

$decoded = $crypto->decrypt($ciphertext);

} catch (InvalidPaddingException $e) {

$decoded = false;

}

} else {

$decoded = preg\_replace('#-.+-|[\r\n]| #', '', $key);

$decoded = preg\_match('#^[a-zA-Z\d/+]\*={0,2}$#', $decoded) ? base64\_decode($decoded) : false;

}

if ($decoded !== false) {

$key = $decoded;

}

$components = array();

if (ord($this->\_string\_shift($key)) != CRYPT\_RSA\_ASN1\_SEQUENCE) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected a SEQUENCE tag');

}

if ($this->\_decodeLength($key) != strlen($key)) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Tag length does not match key length');

}

$tag = ord($this->\_string\_shift($key));

/\* intended for keys for which OpenSSL's asn1parse returns the following:

0:d=0 hl=4 l= 631 cons: SEQUENCE

4:d=1 hl=2 l= 1 prim: INTEGER :00

7:d=1 hl=2 l= 13 cons: SEQUENCE

9:d=2 hl=2 l= 9 prim: OBJECT :rsaEncryption

20:d=2 hl=2 l= 0 prim: NULL

22:d=1 hl=4 l= 609 prim: OCTET STRING \*/

if ($tag == CRYPT\_RSA\_ASN1\_INTEGER && substr($key, 0, 3) == "\x01\x00\x30") {

$this->\_string\_shift($key, 3);

$tag = CRYPT\_RSA\_ASN1\_SEQUENCE;

}

if ($tag == CRYPT\_RSA\_ASN1\_SEQUENCE) {

/\* intended for keys for which OpenSSL's asn1parse returns the following:

0:d=0 hl=4 l= 290 cons: SEQUENCE

4:d=1 hl=2 l= 13 cons: SEQUENCE

6:d=2 hl=2 l= 9 prim: OBJECT :rsaEncryption

17:d=2 hl=2 l= 0 prim: NULL

19:d=1 hl=4 l= 271 prim: BIT STRING \*/

$this->\_string\_shift($key, $this->\_decodeLength($key));

$tag = ord($this->\_string\_shift($key)); // skip over the BIT STRING / OCTET STRING tag

$this->\_decodeLength($key); // skip over the BIT STRING / OCTET STRING length

// "The initial octet shall encode, as an unsigned binary integer wtih bit 1 as the least significant bit, the number of

// unused bits in the final subsequent octet. The number shall be in the range zero to seven."

// -- http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf (section 8.6.2.2)

if ($tag == CRYPT\_RSA\_ASN1\_BITSTRING) {

$this->\_string\_shift($key);

}

if (ord($this->\_string\_shift($key)) != CRYPT\_RSA\_ASN1\_SEQUENCE) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected a second SEQUENCE tag');

}

if ($this->\_decodeLength($key) != strlen($key)) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Tag length and key length should match');

}

$tag = ord($this->\_string\_shift($key));

}

if ($tag != CRYPT\_RSA\_ASN1\_INTEGER) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected an INTEGER tag');

}

$length = $this->\_decodeLength($key);

$temp = $this->\_string\_shift($key, $length);

if (strlen($temp) != 1 || ord($temp) > 2) {

$components['modulus'] = new Math\_BigInteger($temp, 256);

$this->\_string\_shift($key); // skip over CRYPT\_RSA\_ASN1\_INTEGER

$length = $this->\_decodeLength($key);

$components[$type == CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1 ? 'publicExponent' : 'privateExponent'] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

return $components;

}

if (ord($this->\_string\_shift($key)) != CRYPT\_RSA\_ASN1\_INTEGER) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected a second INTEGER tag');

}

$length = $this->\_decodeLength($key);

$components['modulus'] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['publicExponent'] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['privateExponent'] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['primes'] = array(1 => new Math\_BigInteger($this->\_string\_shift($key, $length), 256));

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['primes'][] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['exponents'] = array(1 => new Math\_BigInteger($this->\_string\_shift($key, $length), 256));

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['exponents'][] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['coefficients'] = array(2 => new Math\_BigInteger($this->\_string\_shift($key, $length), 256));

if (!empty($key)) {

if (ord($this->\_string\_shift($key)) != CRYPT\_RSA\_ASN1\_SEQUENCE) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected another SEQUENCE tag');

}

$this->\_decodeLength($key);

while (!empty($key)) {

if (ord($this->\_string\_shift($key)) != CRYPT\_RSA\_ASN1\_SEQUENCE) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PKCS1: Expected yet another SEQUENCE tag');

}

$this->\_decodeLength($key);

$key = substr($key, 1);

$length = $this->\_decodeLength($key);

$components['primes'][] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['exponents'][] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

$this->\_string\_shift($key);

$length = $this->\_decodeLength($key);

$components['coefficients'][] = new Math\_BigInteger($this->\_string\_shift($key, $length), 256);

}

}

return $components;

case CRYPT\_RSA\_PUBLIC\_FORMAT\_OPENSSH:

$parts = explode(' ', $key, 3);

$key = isset($parts[1]) ? base64\_decode($parts[1]) : false;

if ($key === false) {

throw new FormatNotSupportedException('OpenSSH public: Public key not present or badly formated');

}

$comment = isset($parts[2]) ? $parts[2] : false;

$cleanup = substr($key, 0, 11) == "\0\0\0\7ssh-rsa";

if (strlen($key) <= 4) {

throw new FormatNotSupportedException('OpenSSH public: Key data too short');

}

extract(unpack('Nlength', $this->\_string\_shift($key, 4)));

$publicExponent = new Math\_BigInteger($this->\_string\_shift($key, $length), -256);

if (strlen($key) <= 4) {

throw new FormatNotSupportedException('OpenSSH public: public exponent data too short');

}

extract(unpack('Nlength', $this->\_string\_shift($key, 4)));

$modulus = new Math\_BigInteger($this->\_string\_shift($key, $length), -256);

if ($cleanup && strlen($key)) {

if (strlen($key) <= 4) {

throw new FormatNotSupportedException('OpenSSH public: data too short');

}

extract(unpack('Nlength', $this->\_string\_shift($key, 4)));

$realModulus = new Math\_BigInteger($this->\_string\_shift($key, $length), -256);

if (strlen($key)) {

throw new FormatNotSupportedException('OpenSSH public: Junk key data present');

}

return array(

'modulus' => $realModulus,

'publicExponent' => $modulus,

'comment' => $comment

);

} else {

if (strlen($key)) {

throw new FormatNotSupportedException('OpenSSH public: Extra key data present');

}

return array(

'modulus' => $modulus,

'publicExponent' => $publicExponent,

'comment' => $comment

);

}

// http://www.w3.org/TR/xmldsig-core/#sec-RSAKeyValue

// http://en.wikipedia.org/wiki/XML\_Signature

case CRYPT\_RSA\_PRIVATE\_FORMAT\_XML:

case CRYPT\_RSA\_PUBLIC\_FORMAT\_XML:

$this->components = array();

$xml = xml\_parser\_create('UTF-8');

xml\_set\_object($xml, $this);

xml\_set\_element\_handler($xml, '\_start\_element\_handler', '\_stop\_element\_handler');

xml\_set\_character\_data\_handler($xml, '\_data\_handler');

// add <xml></xml> to account for "dangling" tags like <BitStrength>...</BitStrength> that are sometimes added

if (!xml\_parse($xml, '<xml>' . $key . '</xml>')) {

throw new FormatNotSupportedException('XML: Unable to parse key');

}

return isset($this->components['modulus']) && isset($this->components['publicExponent']) ? $this->components : false;

// from PuTTY's SSHPUBK.C

case CRYPT\_RSA\_PRIVATE\_FORMAT\_PUTTY:

$components = array();

$key = preg\_split('#\r\n|\r|\n#', $key);

$type = trim(preg\_replace('#PuTTY-User-Key-File-2: (.+)#', '$1', $key[0]));

if ($type != 'ssh-rsa') {

throw new FormatNotSupportedException('PuTTY: Only ssh-rsa keys are supported');

}

$encryption = trim(preg\_replace('#Encryption: (.+)#', '$1', $key[1]));

$comment = trim(preg\_replace('#Comment: (.+)#', '$1', $key[2]));

$publicLength = trim(preg\_replace('#Public-Lines: (\d+)#', '$1', $key[3]));

$public = base64\_decode(implode('', array\_map('trim', array\_slice($key, 4, $publicLength))));

$public = substr($public, 11);

extract(unpack('Nlength', $this->\_string\_shift($public, 4)));

$components['publicExponent'] = new Math\_BigInteger($this->\_string\_shift($public, $length), -256);

extract(unpack('Nlength', $this->\_string\_shift($public, 4)));

$components['modulus'] = new Math\_BigInteger($this->\_string\_shift($public, $length), -256);

$privateLength = trim(preg\_replace('#Private-Lines: (\d+)#', '$1', $key[$publicLength + 4]));

$private = base64\_decode(implode('', array\_map('trim', array\_slice($key, $publicLength + 5, $privateLength))));

switch ($encryption) {

case 'aes256-cbc':

if (!class\_exists('Crypt\_AES')) {

require\_once('Crypt/AES.php');

}

$symkey = '';

$sequence = 0;

while (strlen($symkey) < 32) {

$temp = pack('Na\*', $sequence++, $this->password);

$symkey.= pack('H\*', sha1($temp));

}

$symkey = substr($symkey, 0, 32);

$crypto = new Crypt\_AES();

}

$is\_encrypted = $encryption != 'none';

if ($is\_encrypted) {

$crypto->setKey($symkey);

$crypto->disablePadding();

try {

$private = $crypto->decrypt($private);

} catch (InvalidPaddingException $e) {

throw new BadPasswordException('Unable to decode key. Likely cause: bad password');

}

}

extract(unpack('Nlength', $this->\_string\_shift($private, 4)));

if (strlen($private) < $length) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PuTTY: length not long enough');

}

$components['privateExponent'] = new Math\_BigInteger($this->\_string\_shift($private, $length), -256);

extract(unpack('Nlength', $this->\_string\_shift($private, 4)));

if (strlen($private) < $length) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PuTTY: length not long enough');

}

$components['primes'] = array(1 => new Math\_BigInteger($this->\_string\_shift($private, $length), -256));

extract(unpack('Nlength', $this->\_string\_shift($private, 4)));

if (strlen($private) < $length) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PuTTY: length not long enough');

}

$components['primes'][] = new Math\_BigInteger($this->\_string\_shift($private, $length), -256);

$temp = $components['primes'][1]->subtract($this->one);

$components['exponents'] = array(1 => $components['publicExponent']->modInverse($temp));

$temp = $components['primes'][2]->subtract($this->one);

$components['exponents'][] = $components['publicExponent']->modInverse($temp);

extract(unpack('Nlength', $this->\_string\_shift($private, 4)));

if (strlen($private) < $length) {

throw $is\_encrypted ?

new BadPasswordException('Unable to decode key. Likely cause: bad password') :

new FormatNotSupportedException('PuTTY: length not long enough');

}

$components['coefficients'] = array(2 => new Math\_BigInteger($this->\_string\_shift($private, $length), -256));

return $components;

}

}

/\*\*

\* Returns the key size

\*

\* More specifically, this returns the size of the modulo in bits.

\*

\* @access public

\* @return Integer

\*/

function getSize()

{

return !isset($this->modulus) ? 0 : strlen($this->modulus->toBits());

}

/\*\*

\* Start Element Handler

\*

\* Called by xml\_set\_element\_handler()

\*

\* @access private

\* @param Resource $parser

\* @param String $name

\* @param Array $attribs

\*/

function \_start\_element\_handler($parser, $name, $attribs)

{

//$name = strtoupper($name);

switch ($name) {

case 'MODULUS':

$this->current = &$this->components['modulus'];

break;

case 'EXPONENT':

$this->current = &$this->components['publicExponent'];

break;

case 'P':

$this->current = &$this->components['primes'][1];

break;

case 'Q':

$this->current = &$this->components['primes'][2];

break;

case 'DP':

$this->current = &$this->components['exponents'][1];

break;

case 'DQ':

$this->current = &$this->components['exponents'][2];

break;

case 'INVERSEQ':

$this->current = &$this->components['coefficients'][2];

break;

case 'D':

$this->current = &$this->components['privateExponent'];

}

$this->current = '';

}

/\*\*

\* Stop Element Handler

\*

\* Called by xml\_set\_element\_handler()

\*

\* @access private

\* @param Resource $parser

\* @param String $name

\*/

function \_stop\_element\_handler($parser, $name)

{

//$name = strtoupper($name);

if ($name == 'RSAKEYVALUE') {

return;

}

$this->current = new Math\_BigInteger(base64\_decode($this->current), 256);

unset($this->current);

}

/\*\*

\* Data Handler

\*

\* Called by xml\_set\_character\_data\_handler()

\*

\* @access private

\* @param Resource $parser

\* @param String $data

\*/

function \_data\_handler($parser, $data)

{

if (!isset($this->current) || is\_object($this->current)) {

return;

}

$this->current.= trim($data);

}

/\*\*

\* Loads a public or private key

\*

\* Returns true on success and false on failure (ie. an incorrect password was provided or the key was malformed)

\*

\* @access public

\* @param String $key

\* @param Integer $type optional

\*/

function loadKey($key, $type = false)

{

if ($type === false) {

$types = array(

CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW,

CRYPT\_RSA\_PRIVATE\_FORMAT\_PKCS1,

CRYPT\_RSA\_PRIVATE\_FORMAT\_XML,

CRYPT\_RSA\_PRIVATE\_FORMAT\_PUTTY,

CRYPT\_RSA\_PUBLIC\_FORMAT\_OPENSSH

);

foreach ($types as $type) {

try {

$components = $this->\_parseKey($key, $type);

break;

} catch (FormatNotSupportedException $e) {

}

}

if (!isset($components)) {

throw new FormatNotSupportedException('Unable to find matching format');

}

} else {

$components = $this->\_parseKey($key, $type);

}

if (isset($components['comment']) && $components['comment'] !== false) {

$this->comment = $components['comment'];

}

$this->modulus = $components['modulus'];

$this->k = strlen($this->modulus->toBytes());

$this->exponent = isset($components['privateExponent']) ? $components['privateExponent'] : $components['publicExponent'];

if (isset($components['primes'])) {

$this->primes = $components['primes'];

$this->exponents = $components['exponents'];

$this->coefficients = $components['coefficients'];

$this->publicExponent = $components['publicExponent'];

} else {

$this->primes = array();

$this->exponents = array();

$this->coefficients = array();

$this->publicExponent = false;

}

return true;

}

/\*\*

\* Sets the password

\*

\* Private keys can be encrypted with a password. To unset the password, pass in the empty string or false.

\* Or rather, pass in $password such that empty($password) && !is\_string($password) is true.

\*

\* @see createKey()

\* @see loadKey()

\* @access public

\* @param String $password

\*/

function setPassword($password = false)

{

$this->password = $password;

}

function setPublicKey($key = false, $type = false)

{

if ($key === false && !empty($this->modulus)) {

$this->publicExponent = $this->exponent;

return true;

}

if ($type === false) {

$types = array(

CRYPT\_RSA\_PUBLIC\_FORMAT\_RAW,

CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1,

CRYPT\_RSA\_PUBLIC\_FORMAT\_XML,

CRYPT\_RSA\_PUBLIC\_FORMAT\_OPENSSH

);

foreach ($types as $type) {

try {

$components = $this->\_parseKey($key, $type);

break;

} catch (FormatNotSupportedException $e) {

}

}

if (!isset($components)) {

throw new FormatNotSupportedException('Unable to find matching format');

}

} else {

$components = $this->\_parseKey($key, $type);

}

if (empty($this->modulus) || !$this->modulus->equals($components['modulus'])) {

$this->modulus = $components['modulus'];

$this->exponent = $this->publicExponent = $components['publicExponent'];

return true;

}

$this->publicExponent = $components['publicExponent'];

return true;

}

function getPublicKey($type = CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1)

{

if (empty($this->modulus) || empty($this->publicExponent)) {

throw new KeyNotLoadedException('No public key has been loaded. Maybe you need to call setPublicKey()?');

}

$oldFormat = $this->publicKeyFormat;

$this->publicKeyFormat = $type;

$temp = $this->\_convertPublicKey($this->modulus, $this->publicExponent);

$this->publicKeyFormat = $oldFormat;

return $temp;

}

function getPrivateKey($type = CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1)

{

if (empty($this->primes)) {

throw new KeyNotLoadedException('No private key has been loaded.');

}

$oldFormat = $this->privateKeyFormat;

$this->privateKeyFormat = $type;

$temp = $this->\_convertPrivateKey($this->modulus, $this->publicExponent, $this->exponent, $this->primes, $this->exponents, $this->coefficients);

$this->privateKeyFormat = $oldFormat;

return $temp;

}

function \_getPrivatePublicKey($mode = CRYPT\_RSA\_PUBLIC\_FORMAT\_PKCS1)

{

if (empty($this->modulus) || empty($this->exponent)) {

throw new KeyNotLoadedException('No private key has been loaded.');

}

$oldFormat = $this->publicKeyFormat;

$this->publicKeyFormat = $mode;

$temp = $this->\_convertPublicKey($this->modulus, $this->exponent);

$this->publicKeyFormat = $oldFormat;

return $temp;

}

function \_\_toString()

{

try {

return $this->getPrivateKey($this->privateKeyFormat);

} catch (KeyNotLoadedException $e) {

return $this->\_getPrivatePublicKey($this->publicKeyFormat);

}

}

function \_generateMinMax($bits)

{

$bytes = $bits >> 3;

$min = str\_repeat(chr(0), $bytes);

$max = str\_repeat(chr(0xFF), $bytes);

$msb = $bits & 7;

if ($msb) {

$min = chr(1 << ($msb - 1)) . $min;

$max = chr((1 << $msb) - 1) . $max;

} else {

$min[0] = chr(0x80);

}

return array(

'min' => new Math\_BigInteger($min, 256),

'max' => new Math\_BigInteger($max, 256)

);

}

function \_decodeLength(&$string)

{

$length = ord($this->\_string\_shift($string));

if ( $length & 0x80 ) { // definite length, long form

$length&= 0x7F;

$temp = $this->\_string\_shift($string, $length);

list(, $length) = unpack('N', substr(str\_pad($temp, 4, chr(0), STR\_PAD\_LEFT), -4));

}

return $length;

}

function \_encodeLength($length)

{

if ($length <= 0x7F) {

return chr($length);

}

$temp = ltrim(pack('N', $length), chr(0));

return pack('Ca\*', 0x80 | strlen($temp), $temp);

}

/\*\*

\* String Shift

\*

\* Inspired by array\_shift

\*

\* @param String $string

\* @param optional Integer $index

\* @return String

\* @access private

\*/

function \_string\_shift(&$string, $index = 1)

{

$substr = substr($string, 0, $index);

$string = substr($string, $index);

return $substr;

}

/\*\*

\* Determines the private key format

\*

\* @see createKey()

\* @access public

\* @param Integer $format

\*/

function setPrivateKeyFormat($format)

{

$this->privateKeyFormat = $format;

}

/\*\*

\* Determines the public key format

\*

\* @see createKey()

\* @access public

\* @param Integer $format

\*/

function setPublicKeyFormat($format)

{

$this->publicKeyFormat = $format;

}

function setHash($hash)

{

// Crypt\_Hash supports algorithms that PKCS#1 doesn't support. md5-96 and sha1-96, for example.

switch ($hash) {

case 'md2':

case 'md5':

case 'sha1':

case 'sha256':

case 'sha384':

case 'sha512':

$this->hash = new Crypt\_Hash($hash);

$this->hashName = $hash;

break;

default:

$this->hash = new Crypt\_Hash('sha1');

$this->hashName = 'sha1';

}

$this->hLen = $this->hash->getLength();

}

function setMGFHash($hash)

{

// Crypt\_Hash supports algorithms that PKCS#1 doesn't support. md5-96 and sha1-96, for example.

switch ($hash) {

case 'md2':

case 'md5':

case 'sha1':

case 'sha256':

case 'sha384':

case 'sha512':

$this->mgfHash = new Crypt\_Hash($hash);

break;

default:

$this->mgfHash = new Crypt\_Hash('sha1');

}

$this->mgfHLen = $this->mgfHash->getLength();

}

function setSaltLength($sLen)

{

$this->sLen = $sLen;

}

function \_i2osp($x, $xLen)

{

$x = $x->toBytes();

if (strlen($x) > $xLen) {

throw new EncodingException('Integer too large');

}

return str\_pad($x, $xLen, chr(0), STR\_PAD\_LEFT);

}

function \_os2ip($x)

{

return new Math\_BigInteger($x, 256);

}

function \_exponentiate($x)

{

if (empty($this->primes) || empty($this->coefficients) || empty($this->exponents)) {

return $x->modPow($this->exponent, $this->modulus);

}

$num\_primes = count($this->primes);

if (defined('CRYPT\_RSA\_DISABLE\_BLINDING')) {

$m\_i = array(

1 => $x->modPow($this->exponents[1], $this->primes[1]),

2 => $x->modPow($this->exponents[2], $this->primes[2])

);

$h = $m\_i[1]->subtract($m\_i[2]);

$h = $h->multiply($this->coefficients[2]);

list(, $h) = $h->divide($this->primes[1]);

$m = $m\_i[2]->add($h->multiply($this->primes[2]));

$r = $this->primes[1];

for ($i = 3; $i <= $num\_primes; $i++) {

$m\_i = $x->modPow($this->exponents[$i], $this->primes[$i]);

$r = $r->multiply($this->primes[$i - 1]);

$h = $m\_i->subtract($m);

$h = $h->multiply($this->coefficients[$i]);

list(, $h) = $h->divide($this->primes[$i]);

$m = $m->add($r->multiply($h));

}

} else {

$smallest = $this->primes[1];

for ($i = 2; $i <= $num\_primes; $i++) {

if ($smallest->compare($this->primes[$i]) > 0) {

$smallest = $this->primes[$i];

}

}

$one = new Math\_BigInteger(1);

$r = $one->random($one, $smallest->subtract($one));

$m\_i = array(

1 => $this->\_blind($x, $r, 1),

2 => $this->\_blind($x, $r, 2)

);

$h = $m\_i[1]->subtract($m\_i[2]);

$h = $h->multiply($this->coefficients[2]);

list(, $h) = $h->divide($this->primes[1]);

$m = $m\_i[2]->add($h->multiply($this->primes[2]));

$r = $this->primes[1];

for ($i = 3; $i <= $num\_primes; $i++) {

$m\_i = $this->\_blind($x, $r, $i);

$r = $r->multiply($this->primes[$i - 1]);

$h = $m\_i->subtract($m);

$h = $h->multiply($this->coefficients[$i]);

list(, $h) = $h->divide($this->primes[$i]);

$m = $m->add($r->multiply($h));

}

}

return $m;

}

function \_blind($x, $r, $i)

{

$x = $x->multiply($r->modPow($this->publicExponent, $this->primes[$i]));

$x = $x->modPow($this->exponents[$i], $this->primes[$i]);

$r = $r->modInverse($this->primes[$i]);

$x = $x->multiply($r);

list(, $x) = $x->divide($this->primes[$i]);

return $x;

}

function \_equals($x, $y)

{

if (strlen($x) != strlen($y)) {

return false;

}

$result = 0;

for ($i = 0; $i < strlen($x); $i++) {

$result |= ord($x[$i]) ^ ord($y[$i]);

}

return $result == 0;

}

function \_rsaep($m)

{

if ($m->compare($this->zero) < 0 || $m->compare($this->modulus) > 0) {

throw new EncryptionException('Message representative out of range');

}

return $this->\_exponentiate($m);

}

function \_rsadp($c)

{

if ($c->compare($this->zero) < 0 || $c->compare($this->modulus) > 0) {

throw new DecryptionException('Ciphertext representative out of range');

}

return $this->\_exponentiate($c);

}

function \_rsasp1($m)

{

if ($m->compare($this->zero) < 0 || $m->compare($this->modulus) > 0) {

throw new SigningException('Message representative out of range');

}

return $this->\_exponentiate($m);

}

function \_rsavp1($s)

{

if ($s->compare($this->zero) < 0 || $s->compare($this->modulus) > 0) {

throw new VerificationException('Signature representative out of range');

return false;

}

return $this->\_exponentiate($s);

}

function \_mgf1($mgfSeed, $maskLen)

{

// if $maskLen would yield strings larger than 4GB, PKCS#1 suggests a "Mask too long" error be output.

$t = '';

$count = ceil($maskLen / $this->mgfHLen);

for ($i = 0; $i < $count; $i++) {

$c = pack('N', $i);

$t.= $this->mgfHash->hash($mgfSeed . $c);

}

return substr($t, 0, $maskLen);

}

function \_rsaes\_oaep\_encrypt($m, $l = '')

{

$mLen = strlen($m);

// Length checking

// if $l is larger than two million terrabytes and you're using sha1, PKCS#1 suggests a "Label too long" error

// be output.

if ($mLen > $this->k - 2 \* $this->hLen - 2) {

throw new EncryptionException('Message too long');

}

// EME-OAEP encoding

$lHash = $this->hash->hash($l);

$ps = str\_repeat(chr(0), $this->k - $mLen - 2 \* $this->hLen - 2);

$db = $lHash . $ps . chr(1) . $m;

$seed = crypt\_random\_string($this->hLen);

$dbMask = $this->\_mgf1($seed, $this->k - $this->hLen - 1);

$maskedDB = $db ^ $dbMask;

$seedMask = $this->\_mgf1($maskedDB, $this->hLen);

$maskedSeed = $seed ^ $seedMask;

$em = chr(0) . $maskedSeed . $maskedDB;

// RSA encryption

$m = $this->\_os2ip($em);

$c = $this->\_rsaep($m);

$c = $this->\_i2osp($c, $this->k);

// Output the ciphertext C

return $c;

}

function \_rsaes\_oaep\_decrypt($c, $l = '')

{

// Length checking

// if $l is larger than two million terrabytes and you're using sha1, PKCS#1 suggests a "Label too long" error

// be output.

if (strlen($c) != $this->k || $this->k < 2 \* $this->hLen + 2) {

throw new DecryptionException('Decryption error');

}

// RSA decryption

$c = $this->\_os2ip($c);

$m = $this->\_rsadp($c);

if ($m === false) {

throw new DecryptionException('Decryption error');

}

$em = $this->\_i2osp($m, $this->k);

// EME-OAEP decoding

$lHash = $this->hash->hash($l);

$y = ord($em[0]);

$maskedSeed = substr($em, 1, $this->hLen);

$maskedDB = substr($em, $this->hLen + 1);

$seedMask = $this->\_mgf1($maskedDB, $this->hLen);

$seed = $maskedSeed ^ $seedMask;

$dbMask = $this->\_mgf1($seed, $this->k - $this->hLen - 1);

$db = $maskedDB ^ $dbMask;

$lHash2 = substr($db, 0, $this->hLen);

$m = substr($db, $this->hLen);

if ($lHash != $lHash2) {

throw new DecryptionException('Decryption error');

}

$m = ltrim($m, chr(0));

if (ord($m[0]) != 1) {

throw new DecryptionException('Decryption error');

}

// Output the message M

return substr($m, 1);

}

function \_rsaes\_pkcs1\_v1\_5\_encrypt($m)

{

$mLen = strlen($m);

// Length checking

if ($mLen > $this->k - 11) {

throw new EncryptionException('Message too long');

}

// EME-PKCS1-v1\_5 encoding

$psLen = $this->k - $mLen - 3;

$ps = '';

while (strlen($ps) != $psLen) {

$temp = crypt\_random\_string($psLen - strlen($ps));

$temp = str\_replace("\x00", '', $temp);

$ps.= $temp;

}

$type = 2;

// see the comments of \_rsaes\_pkcs1\_v1\_5\_decrypt() to understand why this is being done

if (defined('CRYPT\_RSA\_PKCS15\_COMPAT') && (!isset($this->publicExponent) || $this->exponent !== $this->publicExponent)) {

$type = 1;

// "The padding string PS shall consist of k-3-||D|| octets. ... for block type 01, they shall have value FF"

$ps = str\_repeat("\xFF", $psLen);

}

$em = chr(0) . chr($type) . $ps . chr(0) . $m;

// RSA encryption

$m = $this->\_os2ip($em);

$c = $this->\_rsaep($m);

$c = $this->\_i2osp($c, $this->k);

// Output the ciphertext C

return $c;

}

function \_rsaes\_pkcs1\_v1\_5\_decrypt($c)

{

// Length checking

if (strlen($c) != $this->k) { // or if k < 11

throw new DecryptionException('Decryption error');

}

// RSA decryption

$c = $this->\_os2ip($c);

$m = $this->\_rsadp($c);

if ($m === false) {

throw new DecryptionException('Decryption error');

}

$em = $this->\_i2osp($m, $this->k);

// EME-PKCS1-v1\_5 decoding

if (ord($em[0]) != 0 || ord($em[1]) > 2) {

throw new DecryptionException('Decryption error');

}

$ps = substr($em, 2, strpos($em, chr(0), 2) - 2);

$m = substr($em, strlen($ps) + 3);

if (strlen($ps) < 8) {

throw new DecryptionException('Decryption error');

}

// Output M

return $m;

}

/\*\*

\* EMSA-PSS-ENCODE

\*

\* See {@link http://tools.ietf.org/html/rfc3447#section-9.1.1 RFC3447#section-9.1.1}.

\*

\* @access private

\* @param String $m

\* @param Integer $emBits

\*/

function \_emsa\_pss\_encode($m, $emBits)

{

// if $m is larger than two million terrabytes and you're using sha1, PKCS#1 suggests a "Label too long" error

// be output.

$emLen = ($emBits + 1) >> 3; // ie. ceil($emBits / 8)

$sLen = $this->sLen == false ? $this->hLen : $this->sLen;

$mHash = $this->hash->hash($m);

if ($emLen < $this->hLen + $sLen + 2) {

throw new SigningException('Encoding error');

}

$salt = crypt\_random\_string($sLen);

$m2 = "\0\0\0\0\0\0\0\0" . $mHash . $salt;

$h = $this->hash->hash($m2);

$ps = str\_repeat(chr(0), $emLen - $sLen - $this->hLen - 2);

$db = $ps . chr(1) . $salt;

$dbMask = $this->\_mgf1($h, $emLen - $this->hLen - 1);

$maskedDB = $db ^ $dbMask;

$maskedDB[0] = ~chr(0xFF << ($emBits & 7)) & $maskedDB[0];

$em = $maskedDB . $h . chr(0xBC);

return $em;

}

function \_emsa\_pss\_verify($m, $em, $emBits)

{

// if $m is larger than two million terrabytes and you're using sha1, PKCS#1 suggests a "Label too long" error

// be output.

$emLen = ($emBits + 1) >> 3; // ie. ceil($emBits / 8);

$sLen = $this->sLen == false ? $this->hLen : $this->sLen;

$mHash = $this->hash->hash($m);

if ($emLen < $this->hLen + $sLen + 2) {

return false;

}

if ($em[strlen($em) - 1] != chr(0xBC)) {

return false;

}

$maskedDB = substr($em, 0, -$this->hLen - 1);

$h = substr($em, -$this->hLen - 1, $this->hLen);

$temp = chr(0xFF << ($emBits & 7));

if ((~$maskedDB[0] & $temp) != $temp) {

return false;

}

$dbMask = $this->\_mgf1($h, $emLen - $this->hLen - 1);

$db = $maskedDB ^ $dbMask;

$db[0] = ~chr(0xFF << ($emBits & 7)) & $db[0];

$temp = $emLen - $this->hLen - $sLen - 2;

if (substr($db, 0, $temp) != str\_repeat(chr(0), $temp) || ord($db[$temp]) != 1) {

return false;

}

$salt = substr($db, $temp + 1); // should be $sLen long

$m2 = "\0\0\0\0\0\0\0\0" . $mHash . $salt;

$h2 = $this->hash->hash($m2);

return $this->\_equals($h, $h2);

}

function \_rsassa\_pss\_sign($m)

{

// EMSA-PSS encoding

$em = $this->\_emsa\_pss\_encode($m, 8 \* $this->k - 1);

// RSA signature

$m = $this->\_os2ip($em);

$s = $this->\_rsasp1($m);

$s = $this->\_i2osp($s, $this->k);

// Output the signature S

return $s;

}

function \_rsassa\_pss\_verify($m, $s)

{

// Length checking

if (strlen($s) != $this->k) {

throw new VerificationException('Invalid signature');

}

// RSA verification

$modBits = 8 \* $this->k;

$s2 = $this->\_os2ip($s);

$m2 = $this->\_rsavp1($s2);

if ($m2 === false) {

throw new VerificationException('Invalid signature');

}

$em = $this->\_i2osp($m2, $modBits >> 3);

if ($em === false) {

throw new VerificationException('Invalid signature');

}

// EMSA-PSS verification

return $this->\_emsa\_pss\_verify($m, $em, $modBits - 1);

}

/\*\*

\* EMSA-PKCS1-V1\_5-ENCODE

\*

\* See {@link http://tools.ietf.org/html/rfc3447#section-9.2 RFC3447#section-9.2}.

\*

\* @access private

\* @param String $m

\* @param Integer $emLen

\* @return String

\*/

function \_emsa\_pkcs1\_v1\_5\_encode($m, $emLen)

{

$h = $this->hash->hash($m);

if ($h === false) {

return false;

}

// see http://tools.ietf.org/html/rfc3447#page-43

switch ($this->hashName) {

case 'md2':

$t = pack('H\*', '3020300c06082a864886f70d020205000410');

break;

case 'md5':

$t = pack('H\*', '3020300c06082a864886f70d020505000410');

break;

case 'sha1':

$t = pack('H\*', '3021300906052b0e03021a05000414');

break;

case 'sha256':

$t = pack('H\*', '3031300d060960864801650304020105000420');

break;

case 'sha384':

$t = pack('H\*', '3041300d060960864801650304020205000430');

break;

case 'sha512':

$t = pack('H\*', '3051300d060960864801650304020305000440');

}

$t.= $h;

$tLen = strlen($t);

if ($emLen < $tLen + 11) {

throw new EncodingException('Intended encoded message length too short');

}

$ps = str\_repeat(chr(0xFF), $emLen - $tLen - 3);

$em = "\0\1$ps\0$t";

return $em;

}

/\*\*

\* RSASSA-PKCS1-V1\_5-SIGN

\*

\* See {@link http://tools.ietf.org/html/rfc3447#section-8.2.1 RFC3447#section-8.2.1}.

\*

\* @access private

\* @param String $m

\* @return String

\*/

function \_rsassa\_pkcs1\_v1\_5\_sign($m)

{

// EMSA-PKCS1-v1\_5 encoding

$em = $this->\_emsa\_pkcs1\_v1\_5\_encode($m, $this->k);

if ($em === false) {

throw new SigningException('RSA modulus too short');

}

// RSA signature

$m = $this->\_os2ip($em);

$s = $this->\_rsasp1($m);

$s = $this->\_i2osp($s, $this->k);

// Output the signature S

return $s;

}

function \_rsassa\_pkcs1\_v1\_5\_verify($m, $s)

{

// Length checking

if (strlen($s) != $this->k) {

throw new VerificationException('Invalid signature');

return false;

}

// RSA verification

$s = $this->\_os2ip($s);

$m2 = $this->\_rsavp1($s);

if ($m2 === false) {

throw new VerificationException('Invalid signature');

}

$em = $this->\_i2osp($m2, $this->k);

if ($em === false) {

throw new VerificationException('Invalid signature');

}

// EMSA-PKCS1-v1\_5 encoding

$em2 = $this->\_emsa\_pkcs1\_v1\_5\_encode($m, $this->k);

if ($em2 === false) {

throw new VerificationException('RSA modulus too short');

return false;

}

// Compare

return $this->\_equals($em, $em2);

}

/\*\*

\* Set Encryption Mode

\*

\* Valid values include CRYPT\_RSA\_ENCRYPTION\_OAEP and CRYPT\_RSA\_ENCRYPTION\_PKCS1.

\*

\* @access public

\* @param Integer $mode

\*/

function setEncryptionMode($mode)

{

$this->encryptionMode = $mode;

}

function setSignatureMode($mode)

{

$this->signatureMode = $mode;

}

/\*\*

\* Set public key comment.

\*

\* @access public

\* @param String $comment

\*/

function setComment($comment)

{

$this->comment = $comment;

}

function getComment()

{

return $this->comment;

}

function encrypt($plaintext)

{

switch ($this->encryptionMode) {

case CRYPT\_RSA\_ENCRYPTION\_PKCS1:

$length = $this->k - 11;

if ($length <= 0) {

return false;

}

$plaintext = str\_split($plaintext, $length);

$ciphertext = '';

foreach ($plaintext as $m) {

$ciphertext.= $this->\_rsaes\_pkcs1\_v1\_5\_encrypt($m);

}

return $ciphertext;

//case CRYPT\_RSA\_ENCRYPTION\_OAEP:

default:

$length = $this->k - 2 \* $this->hLen - 2;

if ($length <= 0) {

return false;

}

$plaintext = str\_split($plaintext, $length);

$ciphertext = '';

foreach ($plaintext as $m) {

$ciphertext.= $this->\_rsaes\_oaep\_encrypt($m);

}

return $ciphertext;

}

}

function decrypt($ciphertext)

{

if ($this->k <= 0) {

return false;

}

$ciphertext = str\_split($ciphertext, $this->k);

$ciphertext[count($ciphertext) - 1] = str\_pad($ciphertext[count($ciphertext) - 1], $this->k, chr(0), STR\_PAD\_LEFT);

$plaintext = '';

switch ($this->encryptionMode) {

case CRYPT\_RSA\_ENCRYPTION\_PKCS1:

$decrypt = '\_rsaes\_pkcs1\_v1\_5\_decrypt';

break;

//case CRYPT\_RSA\_ENCRYPTION\_OAEP:

default:

$decrypt = '\_rsaes\_oaep\_decrypt';

}

foreach ($ciphertext as $c) {

$temp = $this->$decrypt($c);

if ($temp === false) {

return false;

}

$plaintext.= $temp;

}

return $plaintext;

}

function sign($message)

{

if (empty($this->modulus) || empty($this->exponent)) {

return false;

}

switch ($this->signatureMode) {

case CRYPT\_RSA\_SIGNATURE\_PKCS1:

return $this->\_rsassa\_pkcs1\_v1\_5\_sign($message);

//case CRYPT\_RSA\_SIGNATURE\_PSS:

default:

return $this->\_rsassa\_pss\_sign($message);

}

}

function verify($message, $signature)

{

if (empty($this->modulus) || empty($this->exponent)) {

return false;

}

switch ($this->signatureMode) {

case CRYPT\_RSA\_SIGNATURE\_PKCS1:

return $this->\_rsassa\_pkcs1\_v1\_5\_verify($message, $signature);

//case CRYPT\_RSA\_SIGNATURE\_PSS:

default:

return $this->\_rsassa\_pss\_verify($message, $signature);

}

}

}